

AD-A074 911

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC METHODOLOGY FOR DESIGNING THE ARCHITECTURE OF COMP--ETC(U)
SEP 79 S L HUFF, S E MADNICK
CISR-P010-7906-12

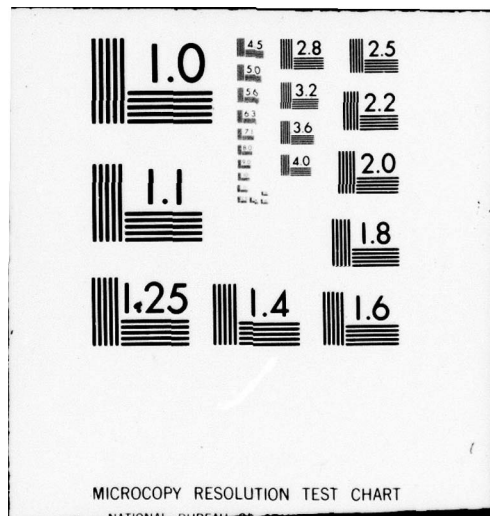
N00039-78-G-0160

NL

UNCLASSIFIED

1 OF 7
ADA
074911





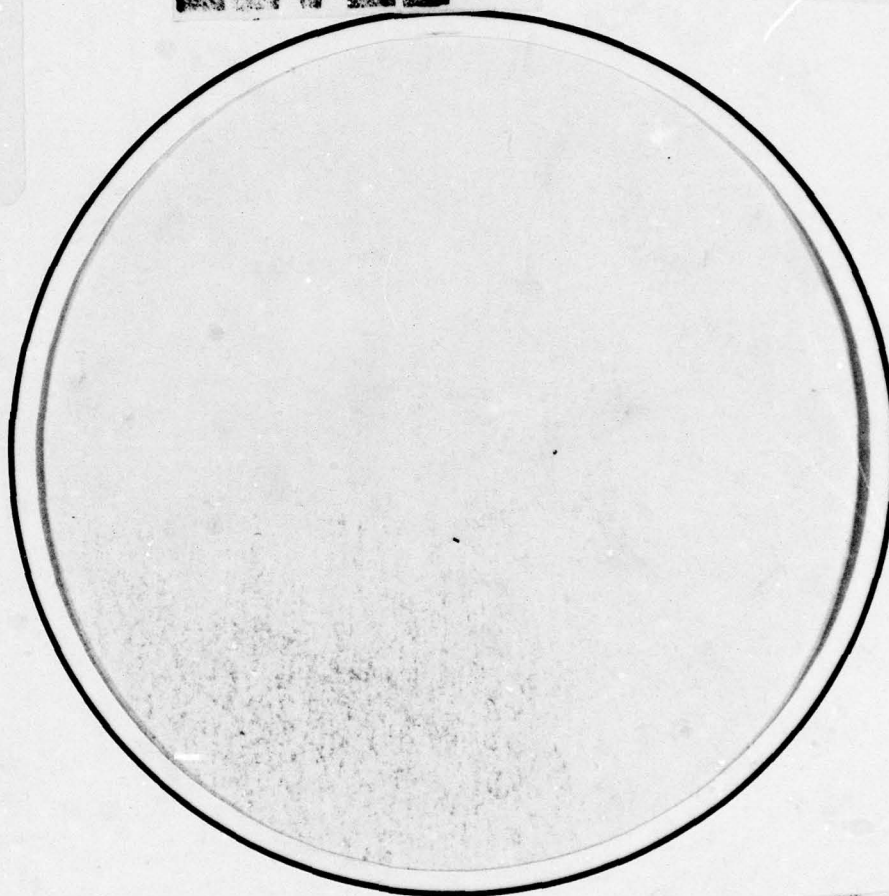
ADA074911



5

LEVEL

RECEIVED
OCT 11 1979
RECEIVED
E



DDC FILE COPY

This document has been approved
for public release and sale; its
distribution is unlimited.

Center for Information Systems Research

Massachusetts Institute of Technology
Alfred P. Sloan School of Management
50 Memorial Drive
Cambridge, Massachusetts, 02139
617 253-1000

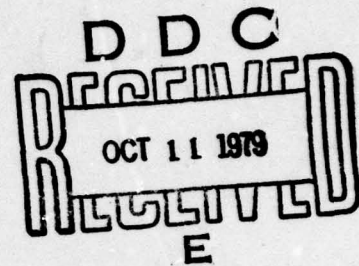
79 10 11 004

Contract Number N00039-78-G-0610 (Order 002)

Internal Report Number P010-7906-12

Deliverable Number 004

(3)



**A SYSTEMATIC METHODOLOGY FOR DESIGNING
THE ARCHITECTURE OF COMPLEX SOFTWARE SYSTEMS**

Technical Report #12

S. L. Huff

September 1979

**Principal Investigator:
Professor S.E. Madnick**

Prepared for:

**Naval Electronic Systems Command
Washington, D.C.**

This document has been approved
for public release and sale; its
distribution is unlimited.

A071748

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (9) Technical Report	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) A Systematic Methodology for Designing The Architecture of Complex Software Systems.		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) (10) S. L. Huff S. E. Madnick		6. PERFORMING ORG. REPORT NUMBER P010-7906-12 ✓
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Information Systems Research Sloan School of Management, M.I.T. Cambridge, Massachusetts 02139		8. CONTRACT OR GRANT NUMBER (15) N00039-78-G-0160 ✓
11. CONTROLLING OFFICE NAME AND ADDRESS Navel Electronic Systems Command		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
12. REPORT DATE (11) Sep 79		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 608		15. SECURITY CLASS. (of this report) Unclassified
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) (14) CISR-P010-7906-12, CISR-TR-12		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software requirements analysis; functional requirements specifications; software architectural design; problem design structuring; graph decomposition.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Complex design problems of all types are characterized by a multitude of competing requirements. Designers of large complex software systems, in particular often find the scope of the problem beyond their cognitive abilities to grasp all at once, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable sub-problems. In software design, this decomposition		

DD FORM 1473
1 JAN 73EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

409590

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

is usually carried out in an ad hoc, intuitive fashion by the system architect, based largely on past experience and his own best judgement. Software research has shown that poor decisions during this preliminary design state, can lead to serious problems, much more difficult to correct, during the later stages of the system development cycle. The aim of the present research is to help develop

The purpose of this research is to contribute to the development of a new software design methodology--the Systematic Design Methodology, or SDM. This methodology is intended to aid the software architect in carrying out preliminary design activities, and in synthesizing a design problem structure for the system under consideration. SDM

Using SDM involves developing a graph model of a system's functional requirements and their implementation interdependencies, then decomposing the graph using various heuristic techniques such as cluster analysis or graph partitioning algorithms. A quantified objective function, derived from the concept of design subproblems possessing high strength, with low coupling between subproblems, is used to locate the best graph decomposition.

As a result of widely varying interpretations of the notion of functional requirement, special guidelines and techniques were developed to aid the user of SDM in creating requirement statements suitable for use in the methodology. In particular, a set of seven statement "templates" was synthesized and tested. The templates were shown to be effective in mapping the information content of typical functional requirement documents in SDM statement form.

A new algorithm for performing hierarchical partitioning of a weighted graph was developed. Tests on a number of graphs showed this algorithm to produce superior decompositions compared to other previously used techniques (principally, clustering algorithms).

A case study applying SDM to a real design problem--the design of a new Institute-wide Budgeting System for M.I.T.--was carried out. This application was conducted jointly with the Budgeting System designers. Anecdotal and other evidence gathered through monitoring their use of, and reactions to SDM attests to the effectiveness of the methodology in assisting software designers in mastering the complexity of real, full-scale architectural design problems.

In order to better understand the methodology's efficacy, the key areas of this potential impact were identified, and some simple models of these impacts upon system development costs were constructed. Finally, an interactive software package was built, using the PL/I programming language, to implement the analytical tools (primarily the graph decomposition algorithms) that were developed in the course of this research.

This report summarizes the work carried out on this project during the past year.

PREFACE

The Center for Information Systems Research (CISR) is a research center of the M.I.T. Sloan School of Management. It consists of a group of management information systems specialists, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means for designing, implementing, and maintaining application software, information systems, and decision support systems.

Within the context of the research effort sponsored by the Naval Electronics Systems Command under contract N00039-78-G-0160, CISR has proposed to conduct basic research on a systematic approach to the early phases of complex systems design. The main goal of this work is the development of a well-defined methodology to fill the gap between system requirements specification and detailed system design.

The research being performed under this contract builds directly upon results stemming from previous research carried out under contract N00039-77-C-0255. The main results of that work include a basic scheme for modelling a set of design problem requirements, techniques for decomposing the requirements set to form a design structure, and guidelines for using the methodology developed from experience gained in testing it on a specific, realistic design problem.

The present study aims to extend and enhance the previous work, primarily through efforts in the following areas:

- 1) additional testing of both the basic methodology, and proposed extensions, through application to other realistic design problems;
- 2) investigation of alternative methods for effectively coupling this methodology together with the preceding and following activities in the systems analysis and design cycle;
- 3) extensions of the earlier representational scheme to allow modelling of additional design-relevant information;
- 4) development of appropriate graph decomposition techniques and software support tools for testing out the proposed extensions.

This document summarizes the work done on this project under task order #002. All of the topics above are addressed in different segments of this report. Many of the results presented here are also covered in earlier reports of this series.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DDC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/ _____	
Availability Codes	
Dist	Avail and/or special
A	

EXECUTIVE SUMMARY

Complex design problems of all types are characterized by a multitude of competing requirements. Designers of large software systems in particular often find the scope of the problem beyond their cognitive abilities to grasp all at once, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable sub-problems. In software design, this decomposition is usually carried out in an ad hoc, intuitive fashion by the system architect, based largely on past experience and his own best judgement. Software research has shown that poor decisions during this, the preliminary design stage, can lead to serious problems, much more difficult to correct, during the later stages of the system development cycle.

The purpose of this research is to contribute to the development of a new software design methodology--the Systematic Design Methodology, or SDM. This methodology is intended to aid the software architect in carrying out preliminary design activities, and in synthesizing a design problem structure for the system under consideration.

Using SDM involves developing a graph model of a system's functional requirements and their implementation interdependencies, then decomposing the graph using various heuristic techniques such as cluster analysis or graph partitioning algorithms. A quantified objective function, derived from the concept of design subproblems possessing high strength, with low coupling between subproblems, is used to locate the best graph decomposition.

As a result of widely varying interpretations of the notion of functional requirement, special guidelines and techniques were developed to aid the user of SDM in creating requirement statements suitable for use in the methodology. In particular, a set of seven statement "templates" was synthesized and tested. The templates were shown to be effective in mapping the information content of typical functional requirement documents in SDM statement form.

A new algorithm for performing hierarchical partitioning of a weighted graph was developed. Tests on a number of graphs showed this algorithm to produce superior decompositions compared to other previously used techniques (principally, clustering algorithms).

A case study applying SDM to a real design problem--the design of a new Institute-wide Budgeting System for M.I.T.--

was carried out. This application was conducted jointly with the Budgeting System designers. Anecdotal and other evidence gathered through monitoring their use of, and reactions to SDM attests to the effectiveness of the methodology in assisting software designers in mastering the complexity of real, full-scale architectural design problems.

In order to better understand the methodology's efficacy, the key areas of its potential impact were identified, and some simple models of these impacts upon system development costs were constructed. Finally, an interactive software package was built, using the PL/I programming language, to implement the analytical tools (primarily the graph decomposition algorithms) that were developed in the course of this research.

This report summarizes the work carried out on this project during the past year.

LIST OF FIGURES

1.1	Increase of software as a proportion of total system cost	20
1.2	The System Development Cycle	23
2.1	A schematic view of the Systematic Design Methodology	54
3.1	PSL object classes and types	78
3.2	PSL relationships and complementary relationships ..	80
3.3	PSL object types and relationship types classified according to system aspect	85
3.4(a)	Diagram for a simple medical monitoring system	87
3.4(b)	First-level PSL description for the MMS	88
3.5	A more detailed description of the MMS	89
3.6	Second-level PSL description of the MMS	92
3.7	Examples of process-oriented requirement statements	102
3.8	A framework for requirement statements	104
3.9	Requirement statement templates	117

3.10	Examples of requirement statement templates	122
4.1	Graph representation of the 22-node design problem	136
4.2	Logical combinations of link weight interpretations	140
4.3	Example of implementation similarity relationship ..	147
4.4	Requirement nodes and interdependency similarities .	150
4.5(a)	A multi-requirement interdependency	151
4.5(b)	A trio of similar pairwise interdependencies	151
4.6	Representation of logical implication between requirement nodes	156
4.7	Representation of hierarchical implication relationships	157
4.8	Representation of implication relationships between implementation nodes	160
4.9	Graph diagram of DBMS design problem using extended model	174
4.10	Basic requirements graph model	180
4.11	Interdependency weights added to model of Figure 4.10	180
4.12	Interdependency similarity information added to model of Figure 4.11	181
4.13	Implication information added to model of Figure 4.12	181
5.1	A simple decomposition	189
5.2(a)	Decomposition of weighted graph	199

5.2(b)	Same decomposition as for Figure 5.2(a), with different weights	199
5.3	Graph decomposition via cluster analysis - the process	201
5.4(a)	An inter-node similarity illustration	203
5.4(b)	A second inter-node similarity illustration	203
5.5	The core set concept	206
5.6	Core sets - the "gravitational" interpretation	208
5.7	Core sets of an weighted graph	210
5.8	Examples illustrating behavior of the similarity measure	214
5.9	More examples illustrating behavior of the similarity measure	215
5.10	Three subsets, ready for next "merge" decision	219
5.11	Single-linkage clustering anomaly	222
5.12	The 22-node DBMS requirements graph	244
5.13	Best decomposition of the unweighted 22-node graph	245
5.14	Best decomposition of the weighted 22-node graph ..	247
5.15	A simple interdependency similarity relationship (ISR)	253
5.16	Representation of the ISR of Figure 5.15	253
5.17(a)	Modification of a three-node subgraph to include an ISR node	257
5.17(b)	Modification of a 4-node subgraph to include an ISR node	257
5.18	The 22-node requirements graph including ISR's	258

5.19	Best cluster decomposition of 22-node graph with ISR's not included	259
5.20	Best cluster decomposition of the 22-node graph using the similarity modification approach for treating ISR's	262
5.21	Best decomposition of the 22-node graph using ISR-node approach to treating the ISR data	264
6.1	A simple weighted graph	281
6.2	(Arbitrary) initial partitioning of graph of Figure 6.1	281
6.3	Depiction of interchange sequence in which early interchanges have negative effect, but a net positive effect eventually accrues	284
6.4(a)	Effect of first pair of interchanges on the graph of Figure 6.2	288
6.4(b)	Resulting situation after interchanges imple- mented	288
6.5	Result after second interchange completed	292
6.6	Best obtained partition during first pass	292
6.7	Depiction of interchange sequence wherein no improvement can be made in the starting partition ..	294
6.8	Graph for second example, showing initial par- tition	296
6.9	Best final partition for second example	298
6.10	Graph for example of Section 6.3, showing dummy nodes and initial partition	306
6.11	Final decomposition of example graph of Section 6.3 after one pass of the interchange algorithm	307
6.12	Initial graph for verification exercise	320

6.13	Graph of Figure 6.12 after nodes x and y have been interchanged	320
6.14	Partitions of the graph from Section 6.2 after one pass of the interchange algorithm, showing subgraph strengths	333
6.15	Decomposition "tree" showing complete decomposition of graph from Section 6.2	377
7.1(a)	Normal pattern for interdependency assessment ...	378
7.1(b)	Inverted pattern for interdependency assessment	378
7.2	Best located decomposition produced by the interchange method on the Budgeting System requirements graph	387
7.3	Relationships among the eleven design subgraphs	389
7.4	Distribution of link total weights	409
7.5	Complete description of the SDM architecture for the new Budgeting System	420
8.1	Graph sketches of relationships for the requirements specification impact model	447
8.2	Graph sketches of relationships for the coordination/communication impact model	453
D.1	Control structure for modules of the SDM analysis package	523

LIST OF TABLES

5.1	Specifications for random and non-random graphs	227
5.2	Relative performance of the four hierarchical clustering techniques	228
5.3	Relative performance summary for the clustering routines	229
5.4	Ranking for clustering routines	229
6.1	Gain matrix for deciding on initial interchange for graph shown in Figure 6.2	287
6.2	Execution trace for interchange example (first pass)	291
6.3	Execution trace for interchange example (second pass, starting with best identified partition from the first pass)	291
6.4	Interchange execution trace for second example	297
6.5	Number of dummy nodes to be added to a graph of n nodes to insure a minimum subgraph size of n nodes	303
6.6	Interchange trace for example of Section 3.3	308
6.7	Initial and final partitions, and interchange trace, for example graph of Section 3.3, with $n = 1$	310
6.8(a)	Interchange trace for example of Section 6.4, without simplifications	325

6.8(b)	Interchange trace for example of Section 6.4, with simplifications	325
6.9	Comparison of interchange and best result obtained using hierarchical clustering	328
6.10	Comparison of the weighted performance of inter- change and the four hierarchical clustering routines	329
7.1	Comparison of results of five decomposition algo- rithms on the Budgeting System requirements graph ...	385
7.2	Subproblem summary description	406
7.3	Summary description of the subproblem interrelation- ships	408
8.1	Typical passes for the requirements analysis and assessments activities	442

TABLE OF CONTENTS

	page
ABSTRACT	2
ACKNOWLEDGMENTS	4
LIST OF FIGURES	5
LIST OF TABLES	10
Chapter	
I. BACKGROUND AND MOTIVATION.	18
Clarification of the problem.	22
Recognition of the Problem By Other Authors.	28
Summary.	30
Outline of the Thesis.	32
II. THE SYSTEMATIC DESIGN METHODOLOGY - PREVIOUS WORK AND ASSOCIATED RESEARCH.	37
Design Theory.	38
Alexander's Research.	40
Software Design Research.	44
Software Architecture Design Research.	48
The SDM Approach.	49
Applications of SDM To Date.	55
Software Architecture Research at Martin Marietta.	58
Specific Research Objectives.	60
Philosophy and Concepts.	60
The SDM Modelling Framework.	62
SDM Decomposition Analysis Techniques.	64
SDM Linkages Within the System Development Cycle.	67
SDM Efficacy.	69
SDM Analysis Package.	70
Testing the SDM.	71
III. REQUIREMENTS STATEMENT CONSTRUCTION - THE SEMANTICS OF REQUIREMENTS.	75
Introduction.	75

Requirements Statement Languages - The Case of	
PSL.	77
The Structure of PSL.	77
Using Objects and Relationships to Create	
PSL Statements.	82
An Illustration of PSL - Part A.	83
An Illustration of PSL - Part B.	86
PSL As a Design Tool.	92
Terminology and Concepts: Some	
Clarifications.	95
Levels of Procedurality.	95
Types of Requirements.	99
Processes and Capabilities.	100
A Framework for Requirements Statements.	103
Expressing Functional Requirements.	111
The Format of Typical Functional	
Specifications.	112
Requirement Statement Templates.	115
Side Effects of Expressing Requirements in	
Template Form.	122
Summary.	128

IV. EXTENSIONS AND IMPROVEMENTS TO THE SDM	
REPRESENTATIONAL MODEL.	129
Introduction.	129
Overview of the Basic Model.	131
Generation of Nodes in the Basic Model.	131
Generation of Links in the Basic Model.	132
An Example.	134
Extensions to the Basic Model.	137
Interdependency Weights.	138
Scaling Problems.	141
Information Linking Implementation Issues.	144
Similarity Links Among Implementation	
Issues.	146
Graph Representation of Implementation	
Issue Commonality.	148
Representation of Implication Information.	151
Logical Implications Between	
Requirements.	151
Logical Implications Between	
Implementation Issues.	158
Logical Implications between R-nodes and	
I-nodes.	159
Application of Extended Model to the 22-node	
DBMS Requirements Set.	162
Sample Set of 22 DBMS Requirements.	165
Requirements Interdependencies and	
Weights.	167
Interdependency Similarity Assessment.	171

Implication Relationships Between Requirements.	172
Implication Relationships Between Implementation Issues.	173
Comments on Assessments.	175
Weight Assessments.	176
Implication Relationship Assessments.	177
Summary.	179

V. GRAPH DECOMPOSITION ANALYSIS TECHNIQUES FOR USE WITH THE EXTENDED SDM MODEL. 183

Introduction.	183
SDM Analysis and Interdependency Weights.	186
Extension to Decomposition Goodness Index.	186
Strength and Coupling - Unweighted Graphs.	187
An Improvement to the Strength Index.	190
Link Weight Information and Similarity.	194
An Example.	198
Extensions to the Basic Similarity Measure.	200
Basic Concepts of Inter-node Similarity.	200
The Core Set Approach.	204
Link Weights and the Core Set Definition.	207
A Further Adjustment.	211
Some Test Cases Using P	213
Clustering Analysis Techniques Using the Extended Model.	217
Four Hierarchical Clustering Techniques.	218
Single Linkage Clustering (HIER1).	221
Complete Linkage Clustering (HIER2).	222
Largest Pre-merge Centroid (HIER3).	223
Largest Post-Merge Centroid (HIER4).	224
Comparative Analysis of Clustering Methods.	225
A "Greedy" Clustering Algorithm.	231
Other Approaches to Graph Decomposition.	234
A Leader Technique.	235
The Bond Energy Approach.	237
Node Tearing Techniques.	240
The Interchange Algorithm.	241
A Case Study Using Interdependency Weight Extensions.	242
Results from the Case Study.	243
SDM Analysis Using Other Model Extensions.	249
Interdependency Similarity Relationships.	251

Modification of Similarity	
Coefficients.	252
Modification of the Graph Structure.	255
An Example of the Use of ISR Data.	256
Summary.	266
VI. SDM DECOMPOSITION ANALYSIS USING THE INTERCHANGE	
ALGORITHM.	269
Introduction.	269
Graph Decomposition in the SDM Context.	270
Shortcomings of Previous SDM Partitioning	
Techniques.	273
The Basic Interchange Algorithm.	278
The Basic Interchange Technique.	279
An Example.	285
A Second Example.	295
Algorithm Extensions - Partition Size Bounding	
and Unequal-sized Subsets.	299
Dummy Nodes.	299
Choice of Dummy Nodes.	301
An Example Using the Dummy Node Technique.	302
Choosing the Number of Dummy Nodes.	309
A Simplified Interchange Algorithm.	311
The Approximate Measure Gain Criterion.	312
Simplifying S	314
Simplifying C	315
Further Simplification of $M(x,y)$	315
A Verification Exercise.	319
Summary of the Simplified Interchange	
Algorithm.	321
Further Efficiency Improvements.	323
Comparison of the Basic and Simplified	
Interchange Algorithms.	324
Comparative Analysis - Interchange versus	
Hierarchical Clustering.	327
Hierarchical Partitioning Using the Interchange	
Technique.	331
The Subgraph Selection Rule.	331
A Stopping Rule for the Master Algorithm.	334
The Master Algorithm.	335
An Example Using the Master Algorithm.	336
Summary.	338
VII. THE USE OF THE SYSTEMATIC DESIGN METHODOLOGY IN THE	
DESIGN OF AN APPLICATION SOFTWARE SYSTEM.	341
Introduction - The Need for SDM Evaluation.	341
Application System Background - The M.I.T.	
Budgeting System.	344
Current M.I.T. Budgeting Environment.	344

The Existing Budgeting System.	348
Top Management.	348
Senior Management.	350
Administrative Management.	352
General Requirements for a New Budgeting System.	354
Preliminary Technical Issues.	355
Support for Top Management.	356
Support for Senior Management.	358
Support for Administrative Management.	360
Support for the Fiscal Planning and Budgeting Office.	362
SDM Analysis of the New Budgeting System.	365
Requirements Preparation.	365
Interdependency Analysis.	370
Some Lessons Learned.	374
Summary.	382
An Architecture for M.I.T.'s New Budgeting System.	383
Decomposition Analysis Results.	384
Analysis of Design Subproblems.	388
Analysis of Subproblem Interrelationships.	407
Summary.	421
VIII. SDM EFFICACY.	428
Introduction.	428
Related Research.	429
The Efficacy Problem.	431
The Case of the New York Times Information Bank.	432
Approaches to the Efficacy Problem.	434
A Predictive Viewpoint of Efficacy.	436
System Specification Impact.	439
System Procedural Development Impact.	448
System Maintenance/Modification Impact.	454
Summary.	464
IX. CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH.	466
Directions for Further Research.	473
REFERENCES	482
Appendix	page
A. COMPLETE LISTING OF PSL STATEMENT TYPES.	492

B.	THE FULL SET OF DBMS REQUIREMENTS USED BY ANDREU.	497
C.	EDITED DBMS REQUIREMENTS TRANSFORMED TO TEMPLATE FORM.	502
D.	SDM ANALYSIS PACKAGE DOCUMENTATION.	510
	More on the MASTER Commands.	512
	INCHCTL (Interchange Control) Commands.	518
	Routines Included in the Analysis Package.	522
	Programs.	524
	Files.	526
E.	TERMINAL EXECUTION TRACE.	527
F.	EXECUTION TRACE OF INTERCHANGE ALGORITHM.	539
G.	EXAMPLE OUTPUT FROM THE IBM 3800 PRINTER.	552
H.	INITIAL BUDGETING SYSTEM FUNCTIONAL REQUIREMENTS AS PREPARED BY THE SYSTEM DESIGNERS.	554
I.	FINAL BUDGETING SYSTEM FUNCTIONAL REQUIREMENTS AS USED IN THE SDM ANALYSIS.	565
J.	FORM USED IN GATHERING INTERDEPENDENCY DATA.	577
K.	BUDGETING SYSTEM REQUIREMENT INTERDEPENDENCIES.	579
L.	REQUIREMENT SUBSETS DERIVED FROM THE BEST SYSTEM DECOMPOSITION.	595
M.	INTERDEPENDENCIES BETWEEN REQUIREMENT SUBSETS IN BEST DECOMPOSITION.	601
	BIOGRAPHICAL NOTE	604

Chapter I

BACKGROUND AND MOTIVATION.

Recent estimates indicate that the design, development, operation, and maintenance, of modern computer-based systems is a twenty billion dollar industry in the United States alone (Dolotta 76), and is growing rapidly. Furthermore, as little as thirty years ago this industry did not exist. The industry's present size, and future growth potential - the substantial amount of resources at stake - has generated much interest in improving both the efficiency and effectiveness with which the save various classes of activities within it are pursued. However, the industry's extremely rapid development to date has often occurred at the expense of carefully planned and rationalized activities. Only very recently, for instance, have system developers generally begun to recognize the wisdom of spending more time and effort planning and designing their systems prior to writing computer programs. (1)

(1) In fairness, the fault is often shared with other organizational managers, who frequently equate code production with progress on a development project.

As well as growth throughout the computer industry, a second important trend is also evident: broadly speaking, the cost of the hardware component of new computer systems has been, and continues to decrease at at a rather rapid pace, on the order of 20 percent per year, whereas the software component continues to increase at nearly as rapid a pace. Viewed over the longer term, the relative importance of the software component of systems today has become pre-eminent, as shown in Figure 1.1.

These circumstances have led to a number of recent studies aimed at identifying problem areas in system development. Certain of these studies have examined the types of design errors typically found in large software systems, and have attempted to trace them back to the design stage in which they were made. The results of these studies indicate that the majority of system errors are made, or have their roots, in the early stages of the system development cycle (Endres 75, Fagan 74, Thayer 75). One particular investigation found that of 54 error types identified during system implementation and maintenance, only 9, or less than 18%, could be attributed to mistakes made during later stages such as coding or testing.

These studies point to the need for improved methods for planning and carrying out the activities that make up the early stages of system development. This thesis is pri-

Percent of
Total Costs

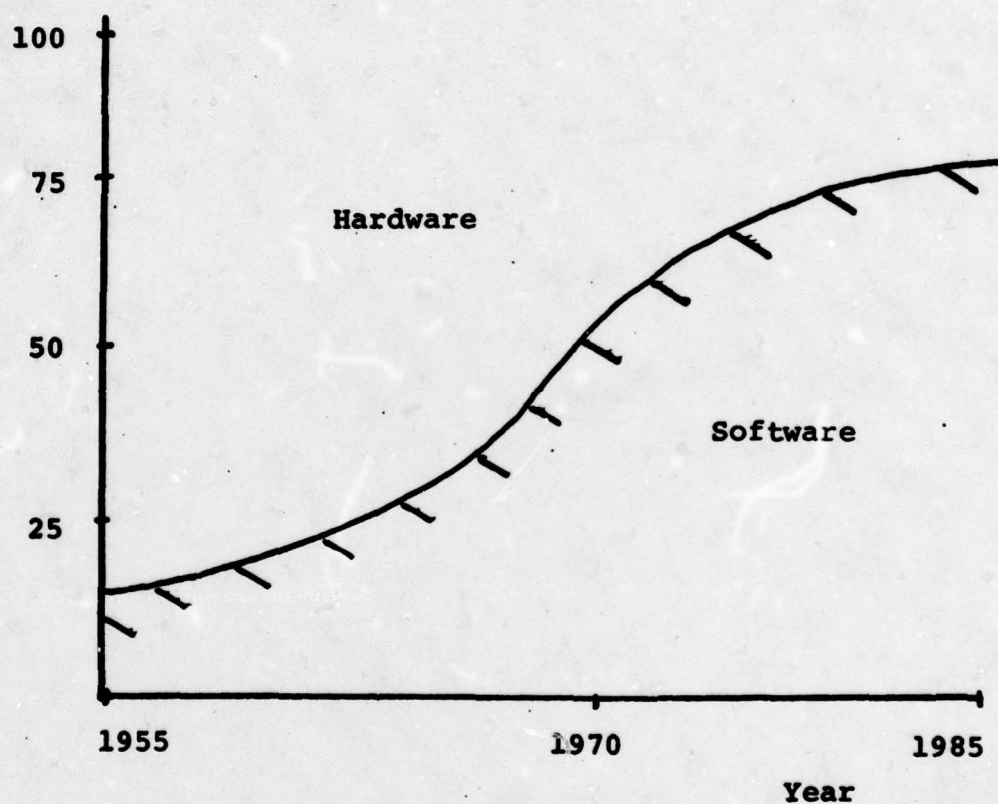


Figure 1.1

Increase of software as a proportion of total system
costs

marily concerned with one aspect of this general need. Specifically, it is concerned with the problem of systematically partitioning the overall problem space faced by a system's designer or design team into an effective set of design sub-problems. A more exact interpretation of the underlined terms above will be given in the next chapter.

1.1 CLARIFICATION OF THE PROBLEM.

One way of making clearer the nature of the problem being addressed is to consider the following scenario. A major system development project has been under way for, say, two years. The team of system analysts has carefully and thoroughly studied the needs of various classes of "users-to-be," and have documented these needs in a system "functional requirements" document. The chief system analyst (often called the system architect) guiding the study of the problem to date now believes he and his staff are ready to begin designing the target system. (2) Moreover, he has organized a number of other small, closely-knit design teams to begin working on various aspects of the system's detailed design. The problem at this point is, how should the overall design problem be partitioned so that it may be "parceled out" to the design teams in the most effective pieces? What, in fact, is meant by an effective partitioning? These questions lie at the heart of this thesis.

The problem may be viewed from a slightly different perspective by considering the well-known system development cycle. While practically every author seems to have his own individual version of the system development cycle, Figure 1.2 is representative. There, the life cycle is shown

(2) Of course in practice there usually does not exist such a clear dividing line between analysis and design. This is an ideal scenario only.

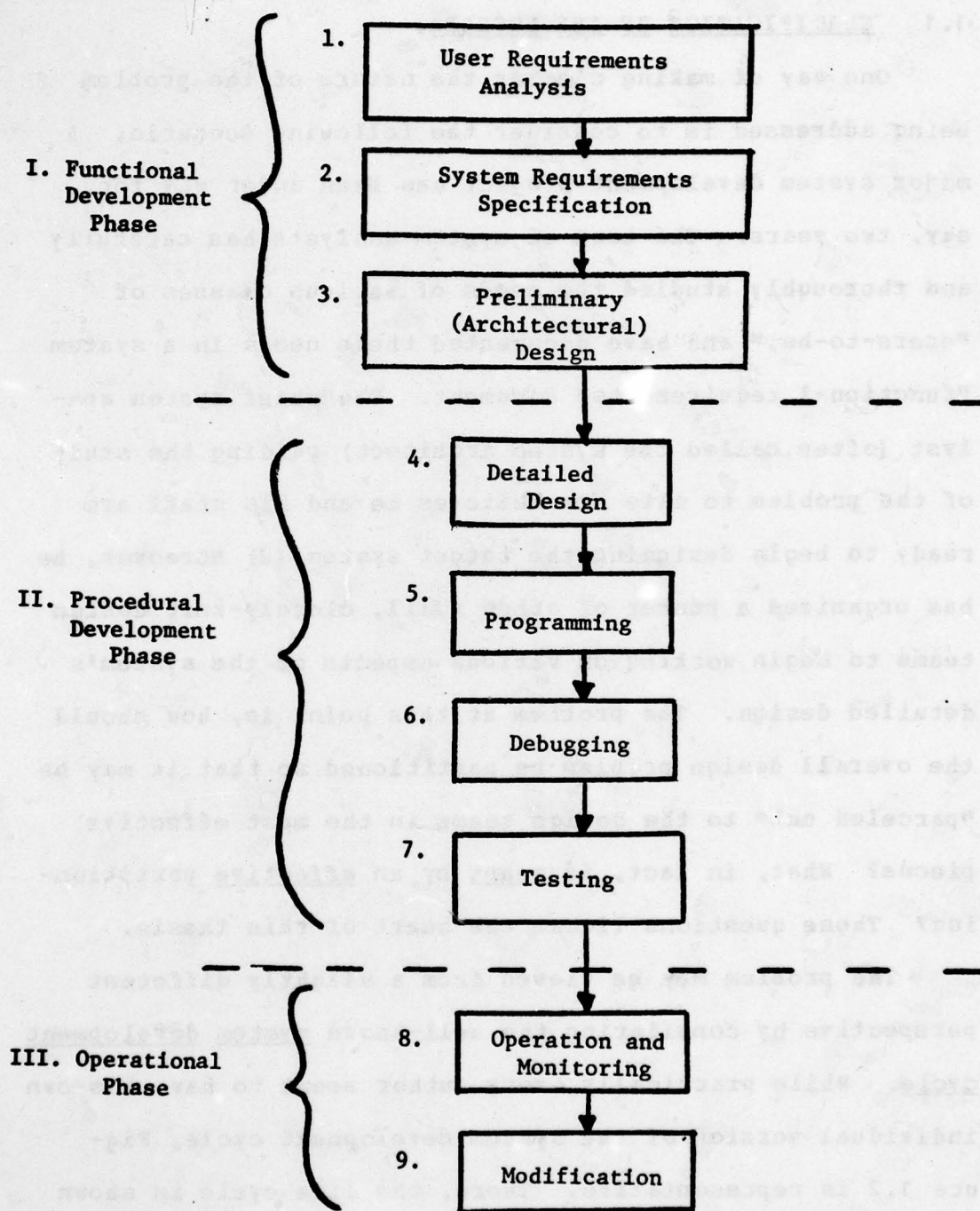


Figure 1.2

The System Development Cycle

to consist of three broad phases, each phase being broken down into more narrowly defined stages. The first, or functional, phase is concerned with:

1. determining what the computer system is intended to do, in the eyes of the user (stage 1);
2. translating the user's functional requirements into system requirements (stage 2);
3. drawing up a high-level preliminary design for the target system (stage 3).

The second, or procedural, phase involves:

1. generating a detailed design for the target system, usually including program module definitions, interface specifications, etc. (stage 4);
2. writing the computer programs, in a suitable programming language, to accomplish the function of each module (stage 5);
3. making sure that the various modules, and the system as a whole, function properly - i.e., the software executes, and executes correctly, as far as can be determined under test conditions (stages 6 and 7).

Finally, the third, or operational, phase involves:

1. putting the system "on the air" - running it under real, as opposed to test, conditions in the user's environment, and monitoring its operation;
2. making modifications, either to fix errors or make requested changes (stage 9).

The activities within the functional development phase are generally less well structured, the objectives and quality measures less well understood, than those in the proce-

dural development or operational phases. Nor have the difficulties associated with the functional development phase gone unnoticed. A few studies have been undertaken to determine how well these tasks are usually executed. For example, Bell and Thayer (Bell and Thayer 76) studied the software requirements specification problem in the context of both a large and a small system. They found a surprisingly large number of errors in this stage of the sample projects they examined - well over one error per page of requirements documentation (the small project gave rise to over 40 pages of such documentation, the larger project over 2500 pages).

Often, the difficulties within the functional development phase begin even earlier, in the user requirements analysis (sometimes referred to as "management information requirements analysis," or "MIRA") stage. In a recent survey, Carter (Carter 75) asked a sample of professional systems analysts, and users, what they felt were the most critical factors contributing to successful development of information systems. Over 120 of these respondents claimed that "the correct identification of management information needs" was the most critical such factor.

What makes system functional development so difficult? One key reason concerns the limited cognitive capabilities of the human brain. It has been observed that most people are able to deal conceptually with only a relatively small number of distinct items of information at a time

(Miller 56). During functional development, the target system must be dealt with as a whole, and typically involves hundreds or thousands of variables, which effectively swamp- ing any single designer's ability to keep mental track of all the pieces. (3)

Within the functional development phase, the task labelled "architectural design" in Figure 1.1 is the central focus of our attention. It is essentially the task of partitioning the problem space so as to be able (conceptually at least) to parcel out sub-problems to design teams, as described above.

A third perspective on the problem being addressed may be obtained by recognizing that the architectural design task is basically the discovery of problem structure. The concept of problem structure may be made clearer by contrasting it with the more familiar notion of system structure. The former is concerned with how different parts of the system interact from a design standpoint, e.g., what parts can be designed independently of others, what parts must be designed at the same time, what tradeoffs should be taken into account when designing somewhat related parts, etc. The latter, in contrast, is concerned with how system

(3) During later stages of the development cycle, the complexity problem is less severe, as any one person is required to deal with a relatively small "chunk" of the total system (e.g., program modules, or subroutines).

parts interact once the system has been designed and put into operation. This involves questions such as what parts communicate with what others, how they do so, etc. Although the design problem structure should, in theory, determine the eventual system structure, the two don't necessarily need to coincide. For instance, it may make good sense to organize several well-defined system functions as separate parts in the final implementation, but if these functions are such that they need to share certain system resources it may be best to organize their design in a common design sub-problem.

Andreu has pointed out (Andreu 78, page 25) that

"traditionally, the 'design problem structure' has been determined by the 'system structure,' rather than the other way around. For instance, it is very common to organize the design of a new system around 'standard' system structures, conceptualized from similar systems designed previously. Although the designer's knowledge of systems similar to the one under design is a valuable input to the design organization activity, it should not be the only one."

With these perspectives of the architectural design problem in mind, we may ask how other researchers view this problem, and whether other effective approaches to solving it have been suggested or studied.

1.2 RECOGNITION OF THE PROBLEM BY OTHER AUTHORS.

The importance of the development of a good system architecture as a blueprint for the following stages in the development cycle is only very recently becoming clearly recognized. White and Booth (White & Booth 76) have expressed concern over "hidden interactions" among system components that might result from overlooking design interdependencies; such interactions are often responsible for difficulties in system maintenance, frequently necessitating major re-design efforts. Belford (Belford et. al. 76) sees the software engineer as "isolated" from the original set of requirements as design implications that stem from original requirements are often overlooked by organizing the target system in ineffective ways that tend to be derived more from previous experience with similar designs than from a systematic investigation of what makes best sense in the present case.

Also, some of the leading authors and researchers in the software design field, while recognizing the importance of the architectural design task, tend to view it as something that "somehow" must take place before various software design methodologies and techniques which they have developed may be applied. Myers, in discussing his Composite/Structured Design Methodology, for instance, writes,

"If the product being developed is a system, rather than a single program, there is another

design process that must occur between the external design process and the use of composite design. This process, called system design, is the decomposition of the system into a set of individual subsystems or individual programs. Although some of the ideas of composite design are appropriate here, and some people have claimed to have used composite design for this process, composite design does not appear to be directly applicable to system design. Therefore, when designing a system, as opposed to an individual program, the designer must first partition the system into distinct subsystems or programs. Then the methodology of composite design can be used to produce the structure of these individual pieces."

Myers also writes (Myers 77, page 158):

"If one is faced with the problem of designing a manufacturing control system, say, one must first determine how this system is to be partitioned into individual programs; then composite design can be applied to design each program. Although some of the ideas can be applied to system partitioning (e.g., maximizing module strength and minimizing inter-module coupling), the decomposition techniques discussed earlier do not seem to be amenable to system partitioning."

In a similar vein, DeWolf (DeWolf 77) has written,

"At the outset, the designer must make some tentative decisions as to the representation of the system and its environment ... Essentially this is a creative activity ..."

These and other authors, therefore, recognize the need to perform the preliminary problem partitioning task, but give no real guidance as to how this activity ought to be carried out. To do that is the cornerstone of this thesis.

1.3 SUMMARY.

The foregoing arguments may be summarized by saying

1. The design and development of computer software systems is an important and challenging problem in modern industrialized societies.
2. There is reasonably wide agreement that many of the difficulties encountered in later stages of the system development cycle stem from poor design decisions made early on.
3. While the need for better managing and structuring of the early (architectural) design phase of system development is becoming more widely recognized, authors and researchers in the software engineering field tend to view this as an unstructured problem for which none of the various software design methodologies in use today are suitable.

This thesis addresses the development and application of a new methodology, the purpose of which is to meet the challenge posed by the third point above. As it is frequently mis-used, the term "methodology" in the above statement bears elaboration. In our view, a methodology has three main components:

1. a philosophy, or guiding set of concepts relating to the purpose and objectives of the methodology;
2. a body of techniques, including such items as mathematical algorithms for solving particular problem types, and guidelines for a step-by-step approach that may be followed in approaching some task; and
3. a set of tools that can help a user of the methodology actually make use of the techniques (for instance, an on-line software package for applying the algorithms to specific problem contexts).

Some other authors sometimes use the term "methodology" when they more properly mean "method," or "technique."

Following early work by Alexander (Alexander 64) in a totally different field, (4) Andreu laid the groundwork for the present research in his doctoral thesis (Andreu 78). These works will be reviewed in the next chapter. The present thesis aims to extend Andreu's earlier work in a number of areas, and to carry out other related studies pertaining to software architecture in general and specifically to the methodology addressed herein: the Systematic Design Methodology ("SDM").

(4) Alexander's work was in the field of conventional architecture, i.e., design of physical structures.

1.4 OUTLINE OF THE THESIS.

In the next chapter, we examine more closely literature in this and related fields. In fact, as intimated earlier, there has been very little prior research, with the exception of Andreu's study, on the software architecture problem per se. However, some useful information may be obtained from other allied fields, such as conventional design theory, software detailed design investigations, etc.; relevant concepts from these areas will be examined. Next, the work of Alexander is examined in some depth, as it forms one of the conceptual cornerstones of the present study. Finally, Andreu's important contributions to the present problem are identified and discussed.

In Chapter 3, we examine a number of issues surrounding the problem of eliciting and expressing the functional requirements for a target software system. Certain important terms, which are frequently used loosely and in ambiguous ways by authors in software engineering are identified, and an attempt is made to clarify them. A simple conceptual model of the role of functional development activities including architectural design, within the overall system development cycle is presented. The role of "requirement statement languages" is discussed, and one specific such language, PSL, is examined in depth. The usefulness of requirement statement languages as software design tools is

addressed. Finally, an approach to generating functional requirement statements in a form suitable for use within this architectural design methodology is given, and examples discussed.

In Chapter 4 we turn our attention to the requirements graph model used by Andreu to represent the key elements of software architecture information: the system's functional requirements, and their interdependencies. The basic question posed in this chapter is, what other kinds of design-relevant information might be knowable and expressible by a system designer, but that are not representable in the original model? For instance, it might be appropriate for designers to be able to express the fact that certain requirements imply the existence of others. A number of such new classes of design information are identified, and representational techniques developed. An example assessment is carried out to illustrate the applicability of the new information types.

In Chapters 5 and 6 we address various aspects of SDH graph decomposition analytics. Chapter 5 includes a number of different, related topics, with its main focus being the extension of the earlier analytical tools to include the most valuable of the model extensions presented in Chapter 4. Most of the attention in Chapter 5 is directed toward factoring the interdependency weight assessment

values into the decomposition analysis activities. Techniques for including interdependency similarity information are also presented and analyzed. New hierarchical clustering algorithms for effecting a graph decomposition are presented, and a comparative analysis among the old and new clustering methods is conducted. Finally, documentation and an example execution trace of the interactive analysis package that has been developed to perform the decomposition analysis is given in the appendices.

A new type of decomposition algorithm, the "interchange" technique, was developed with the SDM decomposition problem specifically in mind. This algorithm is presented in Chapter 6. Also included there is an analysis of ways of simplifying the algorithm's calculations so as to substantially improve its efficiency, together with a comparison of the decomposition effectiveness of this algorithm vis-a-vis the clustering techniques discussed in Chapter 5. The net result is that the interchange algorithm dominates the clustering techniques in all but one case. Finally, a detailed execution trace of the interchange algorithm is given in Appendix F.

Chapter 7 contains a description of the application of the SDM to a real, medium-sized system architectural design problem. The system under design is a new MIT Institute-wide, computer-based budgeting and planning systems. This

is the first application of SDM to the design of an "application" software system,(5) and is also the first time that the key SDM design data has been obtained from the system designers themselves. In previous methodology applications, the role of system designer was "simulated" by the SDM researcher. One of many questions this latter approach left unanswered was how easily a real system designer, initially unfamiliar with the SDM concepts and methods, would be able to "adapt" to the methodology, and what benefits they would see accruing from its use. These and other issues are addressed in Chapter 7. Appendices G through M contain supplementary documentation related to the SDM application exercise.

Chapter 8 focusses on the issue of SDM efficacy. The problem of proving that a methodology such as this one works, that it necessarily, or even probably, leads to a better system design, or that it even generates a net benefit at all, is not a simple one to answer. These questions are addressed, albeit in a preliminary way, in Chapter 8. Specific classes of cost and benefit that are believed to follow from the use of SDM are identified, and approaches to modelling them are explored. While excessive quantification of the suggested relationships would be inappropriate due to

(5) as opposed to a "systems" software system - e.g., an operating system, or a database management system.

the many unproven assumptions that would have to be made, functional-form or graph sketches will be used to illustrate the main ideas.

Finally, Chapter 9 contains conclusions for this thesis, and a discussion of potential fruitful future research directions that could be pursued.

Chapter II

THE SYSTEMATIC DESIGN METHODOLOGY - PREVIOUS WORK AND ASSOCIATED RESEARCH.

The purpose of this chapter is to discuss the specific research upon which this thesis builds. This includes two major pieces of work - Alexander's research, and Andreu's thesis - as well as other miscellaneous work. A number of areas for extension and improvement in Andreu's original study will be identified. These form certain of the objectives of the present work.

As noted in the previous section, the field of software architectural design is almost entirely unresearched. Nevertheless, there are a few allied bodies of research that shed some light on the present problem, and provide directions for pursuit. Related research may be classified into three groups, in increasing order of relevance:

1. research in the general area of design theory;
2. other research in the software engineering field pertaining to software design;
3. research in the software architectural design theory field per se.

We will briefly address each of these three areas, in order.

2.1 DESIGN THEORY.

While software design may be a rather recent object of concerted study, the same cannot be said of design in general. Clearly, practical design problems were being addressed and solved effectively in antiquity (consider the Roman aqueducts, or the Egyptian pyramids). Theoretical tracts on the nature of design go back nearly as far. Most of the early design theorists limited themselves to the enunciation of general principles, formulation of broad concepts, etc. In the past twenty years or so, however, a few design theorists including Archer, Spillers, Jones, Alexander, and Himmelblau have approached the design theory problem from the point of view of constructing useful methodologies (in the sense defined earlier). Their approaches vary widely. Spillers, Archer, and Jones are both "popularizers" and integrators, attempting to construct more or less unified frameworks out of the disparate basic concepts of others. Himmelblau has concentrated on heuristic methods, in an attempt to formulate a general search process for perturbing given designs in order to produce new, better ones. Alexander and others have attempted to represent design problems using various types of graph or network structures, then have defined ways to manipulate these structures to achieve certain objectives such as breaking up complex design problems into more manageable pieces in an intelli-

gent fashion. More will be said about Alexander's work in particular shortly, as it forms part of the foundation for the present study.

The design theory literature is broad and diffuse, and consequently rather difficult to deal with. Most of it is oriented towards classical architecture (the design of structures) or engineering (designing devices). Nor is it a static field. Spillers states in a recent paper, "It is clear that there does not yet exist a unified, coherent theory of design" (Spillers 77). Nevertheless, it contains a rich set of concepts, as well as some methodologies of rather broad scope. To the best of our knowledge, these concepts and methodologies have never been investigated to any significant extent in order to determine what light they may shed on the software design problem.

There is also another, somewhat smaller, literature related to the analysis of structure, as opposed to its design. Representative of this work is Simon's essay on "The Architecture of Complexity" (Simon 70). In this study, Simon addresses the search for common properties among diverse kinds of complex systems. He analyzes four different aspects of complexity, namely,

1. general issues regarding hierarchical form;
2. the evolution of hierarchical and non-hierarchical systems;

3. the concept of nearly decomposable systems;
4. the relationships between complex systems and their descriptions.

A number of Simon's arguments have interesting implications for the software architecture designer.

2.1.1 Alexander's Research.

While the work of Christopher Alexander would properly fall in the previous section, because of its importance and centrality to the present study, it warrants special attention.

Alexander, a design theorist whose main focus has been city design, published the monograph Notes on the Synthesis of Form (Alexander 64) detailing both the philosophy and techniques he had developed in this field. In the book, Alexander recognises the central role of a system's (e.g., a city's) requirements, and their possible interactions,

"A typical design problem ... has requirements which have to be met and there are interactions between the requirements, which makes the requirements hard to meet";

the need for controlling complexity,

"To match the growing complexity of problems, there is a growing body of information and specialist experience. This information is hard to handle; it is widespread, diffuse, unorganized";

and he concludes, as does the software engineering literature, that the intuition of designers is not sufficient to cope with such complexity:

"If we look at the lack of organization and the lack of clarity in the forms around us, it is plain that their design has often taxed their designers' cognitive capacity well beyond its limits ... In this atmosphere, the designer's greatest gift, his intuitive ability to organize physical form, is reduced to nothing by the size of the tasks in front of him."

Alexander then argues that a more systematic approach is required, and that it is not necessary (as some other design theorists contend) to rely entirely on human judgment and intuition in the design activity. Searching for a more systematic approach leads Alexander to think of the design problem as an activity directed to achieve "goodness of fit" between two entities: the form to be designed, and its context. However, he argues, trying to achieve this goodness of fit in a complex environment without the aid of a systematic strategy is futile, because "the number of factors which must fall simultaneously into place is so enormous." Faced with this problem, the designer "tries to break the problem down, and so invents concepts to help himself define which subsets of requirements to deal with independently."

Alexander then points out that typical ways of breaking the problem down, based either on intuition or on standard classifications stemming from traditional disciplines (e.g., economics) tend to fail unless they happen to coincide with the intrinsic structure of the problem at hand. He then

proposes a way to systematize the search for those "subsets of requirements to deal with independently." This is, in effect, a search for problem structure.

Interactions among requirements are at the heart of the methodology proposed by Alexander, and these interactions are defined by him in a way that leaves room for the designer's interpretation:

"We shall say that two requirements interact if and only if the designer can find some reason (or conceptual model) which makes sense to him and tells him why they do so."

Thus, in Alexander's framework, the designer defines the interactions among requirements and attempts to use the information conveyed by them in order to locate subsets of requirements which are as independent as possible of one another. Alexander proposes an approach to formally analyze the structure defined by the requirements and their interactions so as to obtain subsets of strongly interacting requirements which at the same time are as independent of one another as possible. Alexander (and later, Andreu) makes the simplifying assumption that all requirement interactions have the same "strength" or importance. He also distinguishes between "positive" and "negative" requirements, but then proceeds to treat them in exactly the same way for analysis purposes (i.e., simply as "interactions").

With such assumptions, the requirements set may be viewed as an unweighted graph structure, where requirements are graph nodes, and interdependencies are arcs, or links, between the nodes. The problem of identifying subsets of requirements becomes one of graph decomposition. Alexander proposed a simple hill-climbing strategy, attempting to minimize the interactions among subgraphs, to solve the decomposition problem. The resulting subgraphs can be interpreted as "design subproblems" (defined in terms of the associated requirements), that permit a better overall conceptualization of the overall design problem.

With such assumptions, the requirements set may be viewed as an unweighted graph structure, where requirements are graph nodes, and interdependencies are arcs, or links, between the nodes. The problem of identifying subsets of requirements becomes one of graph decomposition. Alexander proposed a simple hill-climbing strategy, attempting to minimize the interactions among subgraphs, to solve the decomposition problem. The resulting subgraphs can be interpreted as "design subproblems" (defined in terms of the associated requirements), that permit a better overall conceptualization of the overall design problem.

2.2 SOFTWARE DESIGN RESEARCH.

A second area of related research is that of software design. Now, programmers and system analysts have been doing software design for over 25 years. However, only during the last few years has significant research interest developed in this field. Wasserman (Wasserman et. al. 78) for instance, observes the following pattern of recent research attention in software engineering:

Years	Focus
pre-1969	Recognition of a "software problem."
1969-1971	First principles; development of broad general concepts.
1972-1973	Structured programming; programming style.
1974-1975	Reliability issues; formal correctness.
1976-1977	Requirements, specification, design; phases prior to coding.
1978-1980	Dispersion, assimilation.

These authors also point out that there is a significant technology transfer delay (sometimes referred to as the "seven year rule") between the development and verification of new ideas in the research community, and their eventual adoption by the software development industry. A recent review (Holton 77), for example, indicated that the basic concepts of structured programming, while enjoying broad

support in the academic and theoretical computer science communities, are being used in a surprisingly small minority of software development organizations.

A major component of the recent wave of software engineering research has been directed at the development of "software design methodologies," (6) including such well-publicized approaches as ISDOS (Teichroew & Hershey 77), SADT (Ross & Schomann 77), HOS (Hamilton & Zeldin 77), SREM (Alford 77; Davis & Vick 77), HIPO (Stay 76), and HDM (Robinson 78). While they vary in details, it is generally the case that these methodologies are targeted toward the task of detailed software design. In fact, this is true (with a few exceptions to be discussed in the next section) of essentially all software design research. Unfortunately, this fact is not necessarily clear from a cursory look at the software engineering literature. The field is still very new, and a standard terminology has not yet settled into place. The same words are frequently used to mean very different things. For example, a recent paper by DeWolf (DeWolf 77) is titled "Requirements Specification and Preliminary Design for Real-time Systems," and describes a technique that employs abstract processes and a data flow scheme to represent the various functional components and

(6) Many of these "methodologies" are more properly termed "methods" or "techniques" according to our earlier definition.

interactions of a proposed system. The point is that DeWolf (in common with most other writers in this field) presumes that the fundamental structure of the problem, and of the proposed solution - that is, the architecture - is given. Judging from the paper's title, however, one would expect it to be about the identification of the need for ("requirements specification"), and the creation of ("preliminary design") this architecture.

The issues of terminology variation and preliminary/detailed design differentiation mentioned above, as well as other important considerations pertaining to the software design literature, are analyzed in greater detail in the next chapter.

A key reason for the importance of the detailed design literature concerns the generality of certain concepts contained therein. A few ideas, pertaining initially to the detailed design problem, have been developed that possess sufficient generality as to have some relevance to preliminary design also. Examples of such general concepts include "information hiding" (Parnas 71), "structured design" (Stevens 75) and "hierarchical decomposition" (Madnick 69; Pyle 72). These concepts are discussed in greater detail in later chapters.

Another important sub-area of related software design research involves requirements specification. Recent

requirement specifications research has centered on the development of techniques, such as formal languages, for capturing the specifications for a target system in a form which meets certain objectives: for instance, functional orientation, completeness, consistency, machine analyzability, etc. Requirement specification research is important for architectural design mainly because the architectural design activity itself begins with statements of system requirements. Therefore, improvements in our ability to express requirements and manage the process of fine tuning a requirements specification should impact the effectiveness with which the software architecture may be created. Unfortunately, it is generally the case that the kind of specifications that researchers have concerned themselves with - their level of detail, degree of procedurality, etc. - are inappropriate for the architectural design task (see Chapter 3). Nevertheless, as in the case of research in detailed software design, some overlap of principles and practices exists. Also, the results of this research may prove useful in the task of building the methodological "bridge" between preliminary and detailed design practice.

2.3 SOFTWARE ARCHITECTURE DESIGN RESEARCH.

The present work stems from the assumption that it is possible (and most desirable) to provide an assisting methodology - a decision support system - to aid the system designer in the task of constructing a system architecture. We call our approach for doing this the Systematic Design Methodology, or "SDM."

The SDM research was initiated by Andreu, who in the course of his doctoral thesis (Andreu 78) formulated some key principles, and developed a first-order implementation, of the methodology. The basic framework developed by Andreu, and the results of some testing of the SDM carried out by Andreu, and by Holden, are reported below.

The present research has as its central objective the extension, strengthening, and further testing of the SDM approach.

One other effort in the software architectural design field with which we are familiar (although it has not been reported in the generally available literature yet) is the work presently ongoing at Martin Marietta Aerospace Corp. A sketch of this study is also given below.

2.3.1 The SDM Approach.

The ultimate objective of the SDM is the identification and description of the macro structure of a target system under design. In practical terms, this may be viewed as the delineation of the major modules, or "sub-problems," into which the target system "ought" to be partitioned (in a sense to be discussed); definition of the function of each identified module; and the identification and functional definition of the interconnections between the modules.

By way of clarification, when we use the term "module" in discussing the SDM, we think of a module as a "subset of system requirements," or "design sub-problem" - not (necessarily) as a program subroutine or collection of such subroutines. Of course, eventually in the design process such subprograms are generated, but the task of laying out pieces of code, subprogram interface conventions, etc., we view as the task of detailed design, not architectural design.

The SDM is driven by the functional requirement specifications for the target system. The initial statement of requirements must be converted into a set of individual, English language statements. For the purposes of the methodology, it is important that the statements exhibit certain characteristics, such as implementation independence, uni-functionality, and design relevance. An approach to the

derivation of appropriate functional requirement statements, based on a set of statement "templates," forms one part of this thesis, and is discussed in the next chapter.

Once the requirement statements have been generated, the system designer must examine each pair of statements in turn, and make a decision as to whether a significant degree of interdependence between the two requirements exists. This he determines by considering how each of the requirements might be implemented in the target system, then asking himself whether he envisions substantial interaction (either interference, or support) between them in the course of performing the implementation. Thus, while the requirement statements themselves are intended to be free of "implementation bias," the assessment of interdependencies demands consideration of alternative modes of implementation. Considerable designer intuition and judgment is required in performing the interdependency assessment. It should be noted that this activity must be performed, in some way, by the designer whether using the SDM or not. Generally this assessment would be carried out in the system architect's head, usually subconsciously, and in a piecemeal, unstructured fashion. The advantage of the SDM approach is that requirements may be treated a pair at a time, thus greatly reducing the complexity of the overall task at the price of additional analysis time.

The resulting information is then represented as a non-directed graph: requirement statements are graph nodes, interdependencies are (unweighted) links. The next step of the methodology is to partition the graph, and by so doing identify the design sub-problems and hence the architectural design.

The graph partitioning phase of the methodology is probably the most "mechanical" step. There are, however, a number of interesting research issues involved here. There are two key aspects to the partitioning task:

1. how to measure the goodness of a given graph partitioning;
2. how to actually locate candidate partitions to be measured.

The first problem centers on exactly what software designers mean by a "good" partitioning for a software system. There is, unfortunately, no broad consensus on this question among software design theoreticians. Different concepts of structure goodness have been proposed, each of which possesses its own positive and negative features. Major drawbacks common to all the proposed criteria, to varying degrees, include

1. the difficulty of quantification;
2. the problem of specifying how to achieve the criterion.

It is common wisdom, for example, that large programs should be constructed in a "modular" fashion, although the writers of those words generally do not specify how to measure how well modularized a given program is, nor do they indicate concrete steps that programmers can follow to ensure their programs meet the criterion of modularity.

In attempting to overcome these drawbacks, SDM incorporates a quantified measure of structure goodness. This measure is based upon the concepts of module strength and inter-module coupling. Alexander (Alexander 64), Stevens (Stevens 75), Myers (Myers 73) and other authors have argued convincingly that a good software design is one that consists of modules that possess high strength, or internal binding, and which simultaneously are weakly interconnected. In SDM, this "strength/coupling" criterion is quantified in the following way. Suppose the graph representation of the target design problem has been decomposed into a set of non-overlapping subgraphs

$$\{G_1, G_2, \dots, G_n\}.$$

Then if S_i = the strength of subgraph G_i , and C_{ij} = the coupling between subgraphs G_i and G_j , we define

$$M = \sum_{i=1}^n S_i - \sum_{i=1}^{n-1} \sum_{j=i+1}^n C_{ij}$$

and use M as a figure of merit for the decomposition. The problem of defining S_i and C_{ij} still remains, and will be discussed in detail in Chapter 5.

The second problem, that of locating the best decomposition of a given graph, is an interesting mathematical problem in its own right. There are basically two strategies that may be used. First, a partitioning strategy seeks to break up the graph into subgraphs until the best decomposition (according to the value of M) is located. Second, a clustering strategy may be followed, wherein a similarity measure between all pairs of graph nodes is defined, then clusters of nodes created by applying one of many possible clustering algorithms to the similarity matrix. Andreu developed particular heuristics using both approaches. Additional work on both partitioning and clustering techniques is presented in this research (Chapters 5 and 6).

Once an agreeable partition has been determined, the resulting sets of requirements are studied and interpreted by the designer as design subproblems. In essence, these design subproblems, together with their interconnections (also derived directly from the graph partition) constitute the preliminary design. The overall SDM procedure is illustrated schematically in Figure 2.1.

Andreu found, in applying SDM to a real set of requirements (for the design of a database management system), that iteration on the overall procedure was of value. The first

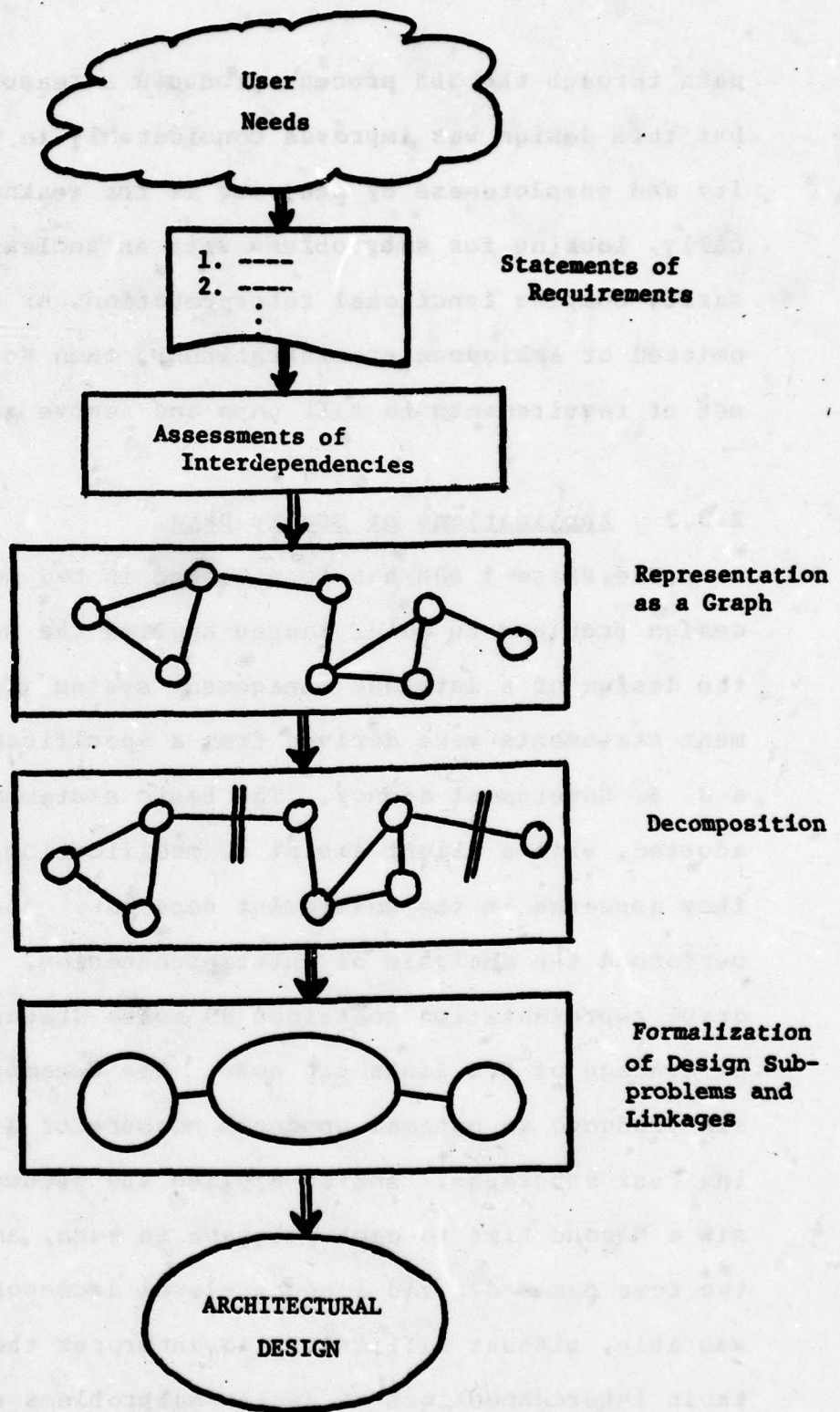


Figure 2.1

A Schematic View of the Systematic Design Methodology

pass through the SDM process produced a reasonable design, but this design was improved considerably in terms of clarity and completeness by studying it for weaknesses (typically, looking for subproblems with an unclear or unnecessarily complex functional interpretation, or for cases of omitted or ambiguous specifications), then "completing" the set of requirements to fill gaps and remove ambiguities.

2.3.2 Applications of SDM To Date.

The Phase-1 SDM has been tested in two non-trivial design problems to date. Andreu applied the methodology to the design of a database management system (DBMS). Requirement statements were derived from a specification issued by a U. S. Government agency. The basic statements were adopted, with a slight amount of modification, directly as they appeared in the government document. Andreu himself performed the analysis of interdependencies. The resulting graph representation contained 89 nodes (requirements) and an average of 5.6 links per node. The decomposition analysis produced an optimal goodness measure of $M=0.806$, yielding four subgraphs. Andreu applied the decomposition analysis a second time to each subgraph in turn, and in two of the four cases derived a second-level decomposition. Andreu was able, without difficulty, to interpret the subgraphs and their interconnections as design subproblems and interactions, hence giving shape to the architecture of the DBMS.

In a separate application test, Holden (Holden 78) applied SDM to the design of a small software operating system. This test differed from the DBMS example in that the target system already existed (as a pedagogical case study in the textbook Operating Systems by Madnick and Donovan) together with substantial descriptive material including the assembly code for the system implementation. Furthermore, a rather carefully thought out design for the system had already been developed and documented (Madnick & Donovan 75).

Holden worked backward to develop the system requirement statements using the written description and code for the operating system, together with informal information gained from discussions with various knowledgeable individuals. A number of iterations were required to build a reasonably clear, consistent, and complete requirement set. Somewhat surprisingly, even though the target system had already been built and documented, the requirement definition task was found to be the most challenging and time-consuming aspect of the SDM test. (7)

(7) Other software engineering researchers have also discovered this, e.g.: "Writing down the requirements turned out to be surprisingly difficult in spite of the availability of a working program and experienced maintenance personnel" (Heninger 79).

Upon application of the decomposition techniques, a decomposition into six subgraphs resulted, with a corresponding goodness measure of $M=1.1$. In two of the six subproblems, a second application of the decomposition analysis resulted in additional partitioning. The subgraphs and their interconnections were again interpreted by the investigator in a straightforward manner to produce the architectural design.

In the case of the operating system, the design produced by SDM resembled the original design in most respects. While this is perhaps not surprising, given the manner in which the requirements were generated, it was encouraging to see that, at least in this case, the SDM appeared to be "stable." A few interesting differences between the original and SDM designs were noted, and were analyzed by Holden. While it was impossible to prove the case conclusively, Holden felt that, in most cases where design differences arose, SDM had produced an alternative that appeared to be as good, or possibly better, a design as the (carefully crafted) original.

It is fair to conclude that the two tests of SDM have been reasonably successful. Much has been learned on the basis of these tests to suggest directions for further research and improvement of SDM (see the sections on "Areas for Further Research" in both (Andreu 78) and (Holden 78)).

The present study is a pursuit of the most promising of these objectives. In Section 3 of this chapter we delineate the specific tasks we plan to address.

2.3.3 Software Architecture Research at Martin Marietta.

A small software research group at Martin Marietta Aerospace Corp., headed by Paul Scheffer, has recently been investigating the software architecture problem from a direction somewhat different from that of the SDM. Scheffer's group has focussed primarily on measurement of the quality of an architecture after it has been created and documented.

The major approach these researchers have pursued to date involves three main activities (Velez & Scheffer 78):

1. documentation of a given software design in the PSL specification language (Teichroew and Hershey 77);
2. translation of the PSL version of the design into a graph framework (PSL "objects" are taken to be graph nodes; PSL "relationships" become graph links);
3. application of alternative decomposition algorithms to the graph so as to determine the quality of the design structure.

In developing the graph analysis techniques and decomposition quality measures, the Martin Marietta group borrowed a number of ideas from the Phase-I SDM, including some of Andreu's clustering heuristics and the basic Phase-I decomposition goodness measure (Andreu 78).

While there is some overlap between the SDM and the Martin Marietta work, it is important to note that the basic orientations of these efforts are quite different. The SDM is fundamentally a designer decision support methodology, and is supposed to play an active role in the development of the architecture for a target software system. The Martin Marietta approach, in contrast, is primarily passive, in that its thrust is simply to quantify "how good" a given architecture is (on an ordinal scale). Before such a measurement can be made, the architecture itself must have been fully constructed and documented in PSL.

2.4 SPECIFIC RESEARCH OBJECTIVES.

In this section we outline the specific directions we intend to pursue in the present research, and the objectives we hope to achieve.

Our overall goal is to make significant extensions and improvements to the Systematic Design Methodology. In pursuing this goal, seven major areas will be investigated, namely,

1. Basic philosophy and concepts;
2. The SDM modelling framework;
3. SDM decomposition analysis techniques;
4. The linkage of the SDM to preceding and following activities within the system development cycle;
5. Efficacy of the SDM;
6. Development of the computer-based analysis package for application of the SDM;
7. Testing the SDM.

Each area is addressed in more depth below.

2.4.1 Philosophy and Concepts.

As mentioned earlier, there is very little literature directly relevant to the software architecture design problem, other than previous work on the SDM. On the other hand, there is a substantial amount of potentially related literature, in areas ranging from general design theory to

software detailed design. Only a relatively small portion of this literature was reviewed by Andreu in his dissertation. It would be worthwhile, therefore, to further explore these related fields to determine whether they contain bodies of concepts, models, frameworks, etc. that might prove of use in better understanding the software architecture problem. The objective of this effort, then, would be simply to discover what, if anything, these related literatures have to offer in defining, clarifying, and solving the problem of software architectural design.

Additional development of the philosophy and concepts relating to software architecture may also derive directly from additional thinking about the nature of the problem, especially when such thinking is carried out in light of experience with the SDM. We pointed out the fact that very basic terminology in the software design area is often used carelessly and ambiguously, for reasons arising primarily from a lack of broadly accepted definitions of key terms, and delineations of central concepts. Some careful classification is definitely required, and would be accomplished by first identifying a core set of terms and concepts upon which the software design field in general, and the SDM in particular, is based; second, discovering and reporting explicit and implicit definitions and meanings for them based on relevant literature; and finally, by suggesting

normative definitions, which would follow from whatever literature consensus may exist, from rational argument, and from the present (SDM) context.

Such a study would also benefit the software design field generally, through additional rationalization and clarification of these key concepts. Put differently, while our objective is to build a good architectural design framework out of concepts in the related fields, our efforts in this direction should "feed back" on these related fields - particularly the software design field - and bring additional structure to them.

2.4.2 The SDM Modelling Framework.

Andreu's initial development of the SDM employed a very simple scheme to model the design problem in terms of an undirected network, or graph. The simplicity of the basic graph model was quite appropriate at that stage, since the entire methodology was new, and parsimony was necessary. However, the main purpose of the graph model is to allow the software architect to express as much information pertaining to the target design problem as possible, within some cost/effectiveness constraint. There is considerable additional design-relevant information, which an average software architect would possess, that is not expressable in the basic graph framework.

The objective of this portion of the research is to

1. identify the additional key classes of design-relevant information that software designers would draw upon (usually intuitively) in constructing a practical architectural design;
2. develop useful extensions to the basic graph model that would serve to effectively represent these additional kinds of information, and
3. to determine which of the extensions identified in point 2 ought to be adopted for the purpose of extending the SDM.

Examples of the kind of model extensions that would be considered include:

1. the expression of the "strength" of interdependencies, which could be modelled as weights on inter-node links corresponding to the associated interdependencies;
2. the relationships that may exist between interdependencies themselves, which could be modelled through the introduction of additional "dummy" nodes in the design graph;
3. additional types of requirement interrelationships, including
 - implication
 - hierarchical
 - concordant vs. discordant.

These and other possible model extensions are analyzed in Chapter 4.

2.4.3 SDM Decomposition Analysis Techniques.

The graph decomposition problem is central to the actual execution of the SDM. From the viewpoint of the SDM itself, the graph decomposition is basically a mechanical task that must be accomplished "somehow." Unfortunately, although graph decomposition has received a fair amount of attention from researchers in other fields, the problem is strongly context dependent, and earlier reported treatments do not exactly "fit" the present context. Of course, there are certain common principles, but implementation details are generally context specific. Andreu, drawing on the common principles, formulated and implemented three main techniques (similarity clustering, "leader group" clustering, and iterative partitioning) for solving the decomposition problem for the basic graph model and basic goodness measure.

Additional study of the decomposition problem, to be pursued in this work, falls into three categories:

1. extension and validation of previously developed algorithms to encompass the extensions to the model and measure (as discussed in the previous section);
2. development of better, or more general-purpose, algorithms;
3. evaluation of competing algorithms.

Each of these points is discussed further below.

As noted, the particular decomposition algorithms required for the SDM are context dependent. Extending the graph representation used within the SDM represents a context alteration, hence impacts the earlier algorithms. These algorithms, if they are to be of use, must be themselves adapted to the new context (if possible), and their effectiveness in that new context tested.

Certain of the algorithms developed by Andreu for the basic model tend to be more useful than others, either for reasons of efficiency (e.g., computer execution time), or effectiveness (e.g., the ability to locate "good" decompositions with regularity). Andreu found the most successful decomposition techniques to be basic clustering algorithms, techniques which group together successively larger "clumps" of nodes in a hierarchical fashion, seeking the particular grouping that gives the largest value of M. The only partitioning technique reported by Andreu, while interesting, proved to be too inefficient in its implementation to be useable for problems of significant size (e.g., over 50 nodes).

One of the technical contributions of the present study is to be the development and testing of a new top-down hierarchical partitioning algorithm, which is well suited to decomposing the particular type of graph being dealt with in this research. This new technique is also applicable to a

fairly broad range of decomposition problems. This algorithm is based on a technique of pairwise interchanging of nodes between two subgraphs so as to continuously improve an objective function (in this case, the extended measure H). Additional details regarding the algorithm and tests that have been carried out with it are given in Chapter 6.

The necessity for having more than one decomposition algorithm centers on the heuristic nature of the algorithms. While the graph decomposition problem can be formulated rather easily as a nonlinear integer program, it is far too large a formulation to be solvable for even relatively small cases (e.g., 30 nodes) in a reasonable amount of time. Hence heuristic approaches - much faster, but not (necessarily) optimal - are required. The confidence that can be placed in the result produced by such heuristics increases if we have a variety of different approaches that can be applied to a given problem. The analysis package being developed (see Section Appendix D) includes a selection of bottom-up clustering techniques together with the new top-down partitioning technique - two fundamentally different approaches to the decomposition problem, which provide the capability for performing a multi-pronged attack on the problem.

While it is useful to have such a selection of techniques available, it may in fact be the case that one of them

consistently dominates the others, or that certain of the techniques consistently produce better results in certain kinds of contexts. It would therefore be beneficial to perform a set of tests of all the decomposition techniques being considered, to determine whether such dominance or contingency situations do in fact exist for the type of graphs typical of the software architecture problem. Such a set of tests has been carried out using the analysis package, and the results are presented in Chapters 5 and 6.

2.4.4 SDM Linkages Within the System Development Cycle.

The SDM is a methodology for developing the architecture of complex software systems. As such, it "fits" within the overall system development cycle between the requirements specification stage and the detailed design stage - see Figure 1.2. In order to further improve the overall effectiveness of the SDM, consideration should be given to ways in which the architectural design stage may be linked together with the preceding (requirements specification) stage and the following (detailed design) stage.

The question of linking stages 2 and 3 is addressed in Chapter 3. The focus of the work reported there is a proposed method for guiding the formulation of functional requirement statements that may be used in constructing the requirements graph in the architectural design stage. A set

of requirement statement "templates" has been developed to assist a systems architect in mapping functional requirement descriptions into a set of requirement statements suitable for use within the Systematic Design Methodology. The template approach is not intended as a way to automate, or even mechanize, the creation of SDM requirements (a task which is surely impossible at this stage in the development cycle). Rather, it is intended simply to structure and guide the thought processes involved in carrying out part of the architectural design activity.

In moving from stage 3 to stage 4, the system designer crosses the boundary from functional development to procedural development (see again Figure 1.2). Essentially, he moves from thinking about what functions are to be provided and which groups of functions are closely related from a design standpoint, to thinking about how he is going to go about providing those functions in the target system. In other words, he shifts from a functional view of development to a procedural, or process-oriented, view. Beyond suggesting a best way to partition the system requirements for follow-on design purposes, SDM itself does not have anything directly to say about how the various functions contained in the requirement statements are to be embodied in the system. It does, however, impact procedural development indirectly. First, conceptual implementation models (discussed more tho-

roughly in Chapter 7) must be created by the system architect in the course of carrying out SDM's interdependency analysis. These models, which are recorded in the interdependency statement descriptions, represent an initial thinking-through of the implementation implications of each requirement statement. Second, interdependency analysis also generates certain "spin-off" effects, essentially helping the designers see more clearly relationships among various implementation schemes, which in turn may lead to better system design concepts (examples of this phenomenon are given in Chapter 7).

While these indirect effects do serve to better link stages 3 and 4 than would be the case without SDM, more work remains to be done on this aspect of the methodology.

2.4.5 SDM Efficacy.

One of the difficulties associated with methodology development research such as this work is the question of ultimate potential and actual value. In the present case, there is an underlying assumption that good software architecture has a favorable impact on the life cycle costs of the system development project. Furthermore, it is assumed that the improvement in lifetime benefits would more than outweigh the additional costs of actually using the SDM in the first place (including costs such as training, additional project duration, staff time, etc.).

The objective of this phase of the present research is to provide a basic analysis of the categories of costs and benefits that will help answer the above questions. Certain relationships underlying the analysis are obtained from the system development cost-modelling literature. The results of this study are given in Chapter 8.

2.4.6 SDM Analysis Package.

In order to carry out tests on various parts of the SDM, an on-line software package has been developed. Using an earlier package developed by Andreu on the PRIME-300 computer (in FORTRAN) as a model, an updated and enhanced version has been developed on the IBM/370 computer in the PL/I programming language. The earlier FORTRAN package was developed to help perform the decomposition analysis of graph structures from the basic model (i.e., undirected graphs with unweighted links). Major extensions to the original package include:

1. the capability to enter, store, manipulate, and alter requirement statements and interdependency information;
2. the capability to derive a weighted graph structure for an extended model representation automatically, from the stored requirements and interdependency information;
3. decomposition analysis tools to deal with graphs in the extended format;

4. decomposition tools to realize new algorithms developed in this research;
5. output facilities to simplify the task of relating graph decompositions to the original formulations in terms of requirements and interdependencies.

PL/I is a significantly better vehicle for development of such a package than is FORTRAN, as its character and based variable data types are especially well suited to handling textual information and graph representations, respectively. Also, the VM/370 CMS file system and executive procedures are powerful tools for the development of such software. Finally, the very large virtual memory available through CMS effectively removes storage constraints that would otherwise hamper the development of this kind of package. Applications of the analysis package are reported in Chapters 5, 6, and 7. Various documentation and example traces are included in Appendices D, E, F, L, and M.

2.4.7 Testing the SDM.

An important part of methodology development is testing within real application situations. In the case of the SDM, testing presents some particularly difficult issues, the main one being the magnitude of time and effort required to effect a realistic test. The difficulty lies in the fact that the SDM is specifically oriented towards large-scale, complex system development efforts (for example, the devel-

opment of a new operating system, an electronic switching system, or a large hospital patient monitoring system). Systems such as these typically require many man-years of effort to design, build, test, and install, and are orders of magnitude too large for one person to address in SDM testing. Even if sheer size was not a problem, the detailed knowledge required concerning the specific application area necessary to fully comprehend the system's requirements and their possible implementation alternatives presents a second major difficulty.

There are, however, ways to proceed with SDM testing. One approach would be to test the methodology against a requirements set that is small enough to be dealt with by one investigator (possibly with the assistance of other interested parties - e.g., master's students). In this case, some assumptions must be made regarding the appropriateness of scaling up the test results to realistic case sizes.

A second approach is the "back-to-front" technique similar to that employed by Holden (Holden 78). With this approach, the investigator would first locate a well-documented, completely developed medium-size system, then work backwards using the system's documentation to develop functional requirement statements for it. The SDM would then be applied to develop a system architecture, which could be

compared to the actual system architecture as per the documentation. Differences would be analyzed, possibly with the aid of the original designers of the system, to attempt to determine the SDM's effectiveness, shortcomings, etc. This approach would be even more effective if the documentation for the system being studied included the original requirements, since this would allow the investigators to avoid having to generate artificial requirements, and would also save testing time.

A third alternative is a real-world test. This approach would require an agreement with an organization currently facing a medium- or large-scale design and development task. The researchers would act as trainers/advisors/observers, with the firm's systems staff actually applying the methodology to their own particular problem. This kind of test would appear to be most promising in terms of identifying the real strengths and weaknesses of the SDM, but also has the most associated difficulties, including the necessity of locating an organization willing to participate and risk the expenditure of some resources on such a test, the difficulty of adequately monitoring and controlling the test, etc.

Following discussions with a number of organizations (primarily Center for Information Systems Research corporate sponsors), it was decided that MIT's own Business Systems

Development group, would be the most suitable organization within which to carry out a test evaluation of this type. The reasons behind this decision, the nature of the arrangements made, the work actually carried out with this group, and the lessons learned are all reported in Chapter 7.

The following six chapters report the results obtained in our investigations of each of the foregoing research objectives.

Chapter III

REQUIREMENTS STATEMENT CONSTRUCTION - THE SEMANTICS OF REQUIREMENTS.

3.1 INTRODUCTION.

In this chapter we address the transition from Stage 1, through Stage 2, to Stage 3 in the System Development Cycle (Figure 1.2). Specifically, we examine the need to capture user-level functional requirements in a form appropriate for follow-on SDM analysis (interdependency assessment, design structure interpretation, etc. - see Figure 2.1). A number of approaches to requirements expression have emerged over the past few years, and our first thought in the area was that one of the well-documented "requirement statement languages" ("RSL's") might prove suitable, perhaps with some modifications, to our needs. In the course of studying this possibility we were led to examine some of these languages, and to assess their nature and functioning with respect to SDM. We were further led to make some general observations regarding ambiguous terminology that has grown up around these RSL's, and around system requirements specification in general, and regarding the appropriate role of RSL's in the System Development Cycle.

We begin this chapter by taking a fairly close look at a particular RSL, one which has served as a cornerstone for much of the research in the requirements statement area. This is the Problem Statement Language, PSL.

We then turn to a brief exploration of the important ambiguities and mis-uses that are frequently encountered in the literature in this area, and attempt to provide some clarifications for them. This leads us to put forth a simple conceptual framework that we have found useful in thinking about requirements, requirement specifications, and requirements languages within the System Development Cycle.

Finally, having concluded that PSL and other similar RSL's are not appropriate for expressing SDM requirement statements, we present a new approach. We term this approach the template technique, as it is based on a set of seven basic "requirement statement templates." Each template corresponds to a general category of statement type. Use of the templates leads the user toward expression of functional requirements in the form of statements that meet certain key criteria for SDM, as discussed in Section 5 of the chapter.

The template technique is illustrated by applying it to the set of DBMS requirements used by Andreu in his SDM application study (Andreu 78). Appendices B and C of the thesis contain the original and edited requirement statements.

3.2 REQUIREMENTS STATEMENT LANGUAGES - THE CASE OF PSL.

Recent years have witnessed a number of attempts to apply computers to the problems inherent in designing and building software systems. One relatively well-known approach is the ISDOS (Information System Design Optimization System) project, which was begun at the University of Michigan in 1969, and is ongoing (Teichroew 71). Out of this project has come the PSL/PSA system, consisting of a formal language - PSL, or "Problem Statement Language" - for specifying a system's functional requirements, and a software support package - PSA, or "Problem Statement Analyzer" - for performing certain kinds of analysis on a set of machine-readable PSL statements, generating various reports from a set of such statements, etc. Our discussion here will focus almost entirely on PSL. For further information regarding the nature and capabilities of both PSL and PSA, see (Teichroew and Bastarache 77).

3.2.1 The Structure of PSL.

The basic model underlying PSL is quite simple. PSL recognizes two kinds of things: objects, and relationships. Objects are "things," such as data elements, logical collections of data, processes, etc. Each PSL object is given a unique name, and is classified as one of 22 possible object types (see Figure 3.1). These 22 object types may be

Classes of Object Types**Object Type within Class**

Interface	INTERFACE
Target System	
Collections of Information	INPUT,OUTPUT,ENTITY
Collections of Instances	SET
Relationships among collections of Information	RELATION
Data Definition	GROUP
Data Derivation	PROCESS
Size and Volume	INTERVAL,SYSTEM-PARAMETER
Dynamic Behavior	EVENT,CONDITION
Project Management	PROBLEM-DEFINER,MAILBOX
Properties	SYNONYM,KEYWORD,ATTRIBUTE, ATTRIBUTE-VALUE,MEMO,SOURCE, SECURITY

Figure 3.1**PSL Object Classes and Types**

grouped into four object classes: interface objects, target system objects, project management objects, and property objects. Interface objects (encompassing one object type) are used to describe the interface between the target system

and its environment. Target system objects (12 object types) together with property objects (7 object types) are used to describe the target system. Project management objects (2 object types) are used to help document the organizational and project control aspects of the system development process.

The other top-level concept in PSL is that of relationship. Relationships are used to state ways in which PSL objects are related to each other. PSL relationships may be likened to "verbs" which, together with PSL objects, serve to generate "sentences," or PSL statements. There are 58 different relationship types included in PSL, although many of these are "inverse pairs" (e.g., the pair RECEIVES, and RECEIVED BY). If we count each such pair as a single "distinct" relationship, the total number is reduced to 31. The various PSL relationships and complementary relationships are listed in Figure 3.2.

PSL semantics requires that only specific types of relationships may be used to interconnect any given pair of objects. For example, the object types ENTITY and PROCESS may legally be interconnected by certain relationships (e.g., process USES entity, or process UPDATES entity), but there are no legal PSL relationships that may interconnect object types ENTITY and INPUT. For a full discussion of PSL semantics, see (Teichroew and Bastarache 77).

Relationship**Associated Complementary
Relationship**

ASSOCIATED	ASSOCIATED-DATA
ATTRIBUTES	--
BECOMING	WHEN
CARDINALITY	--
CONNECTIVITY	--
CONTAINED	CONSISTS
DERIVED	DERIVES
GENERATED	GENERATES
HAPPENS	--
IDENTIFIED	IDENTIFIES
INCEPTION	INCEPTION-CAUSES
KEYWORD	APPLIES
MAILBOX	APPLIES
MAINTAINED	MAINTAINS
PART	SUBPARTS
RECEIVED	RECEIVES
RELATED	BETWEEN
RESPONSIBLE-INTERFACE	RESPONSIBLE
RESPONSIBLE-PROBLEM-DEFINER	RESPONSIBLE
SECURITY	APPLIES
SEE-MEMO	APPLIES
SOURCE	APPLIES
SUBSET	SUBSETS
SUBSETTING-CRITERIA	SUBSETTING-CRITERION
SYNONYM	DESIGNATE
TERMINATION	TERMINATION-CAUSES
TRIGGERED	TRIGGERS
UPDATED	UPDATES
USED	USES
UTILIZED	UTILIZES
VALUES	--

Figure 3.2**PSL Relationships and Complementary Relationships.**

As PSL object types are grouped into object classes, so are relationship types grouped into classes of relationships. Eight relationship classes are defined in PSL, as a function of system "aspect." These eight classes are:

1. system flow
2. system structure
3. data structure
4. data derivation
5. system size
6. system dynamics
7. project management
8. system properties.

PSL objects may also be re-classified according to this scheme. Figure 3.3 shows the classification of both objects and relationships according to system aspect. In cases of complementary relationship pairs, only one is shown.

Information which is needed to describe an object, and which cannot be specified using one or more relationships, can be included in a narrative description called a "comment entry." A number of different types of comment entries may be defined, depending on the type of object to which they pertain. These comment entries are shown starred in Figure 3.3. There are also certain other comment entries that

may be used in every category; these entries are not shown in the figure.

3.2.2 Using Objects and Relationships to Create PSL Statements.

PSL statements have the general form

<object name> <relationship> <object name(s)>.

Typical examples are:

1. payroll-process RECEIVES employee-work-data.

Here, payroll-process would be the name of a PROCESS object, and employee-work-data an INTERFACE object. RECEIVES is a system flow relationship. Note that an equivalent statement would be

employee-work-data IS RECEIVED BY payroll-process.

These two statements express complementary relationships, and are equivalent, both logically and semantically, within PSL.

2. payroll-process SUBPARTS ARE payroll-data-read, pay-calculation, check-print.

This statement describes a hierarchical structure of processes. The right-hand side of the relationship consists of a list of three PROCESS objects. SUBPARTS ARE is a system structure relationship. Note that PSL can only describe hierarchically organized structures of data or objects.

3. net-pay VALUES ARE 0 THRU 2000.

This example illustrates a somewhat different type of statement. Here, net-pay is an ELEMENT object (elementary data type), and the "relationship" is an expression of a range of values that net-pay may legally assume.

If we avoid double-counting statement types such as RECEIVES and IS RECEIVED BY as in the first example above, and omit system property and project management statements (these provide additional detail in a problem statement, but are in the nature of comment entries, as they are not analyzed by PSA), then there are 39 different PSL statement types. These are given in Appendix A in full detail, grouped according to system aspect, and are summarized in Figure 3.3. The system property and project management statement types (see Appendix A) add another 11 to the list of statement types given in Figure 3.3. However, as these types serve only to add peripheral commentary to a PSL description, they are not strictly necessary in stating the logic of a processing problem.

3.2.3 An Illustration of PSL - Part A.

Using PSL involves four main activities:

(a) identifying and naming objects, and assigning a unique type to each; (b) determining the relationships among the objects; (c) writing PSL statements to describe the target system; (d) writing comment entries to express any necessary information which cannot be expressed in the formal syntax.

In order to illustrate both the general idea of PSL, and some of the detailed syntax, we consider here a specific

PSL OBJECT TYPES AND RELATIONSHIP TYPES
CLASSIFIED ACCORDING TO SYSTEM ASPECT.

<u>System Aspect</u>	<u>Object Types</u>	<u>Distinct Statement Types</u>	<u>No. Statement Types</u>
System Flow	PROCESS INTERFACE INPUT OUTPUT	RECEIVES, GENERATES RESPONSIBLE FOR	3
System Structure	PROCESS INTERFACE INPUT,OUTPUT SET,ENTITY ELEMENT,GROUP INTERVAL	PART OF CONTAINED IN SUBSET OF UTILIZED BY CONSISTS OF SUBSETTING-CRITERIA ARE	6
Data Structure	INPUT,OUTPUT SET,ENTITY GROUP,ELEMENT RELATION	IDENTIFIED BY RELATED TO...VIA CONSISTS OF CONTAINED IN BETWEEN...AND ASSOCIATED WITH	6
Data Derivation	PROCESS INPUT,OUTPUT SET,ENTITY ELEMENT,GROUP RELATION	USED BY USED BY...TO DERIVE USED BY...TO UPDATE DERIVED BY DERIVED BY...USING UPDATED BY UPDATED BY...USING MAINTAINS,PROCEDURE DERIVATION	10
System Size	PROCESS,EVENT SET,ENTITY ELEMENT RELATION INTERVAL SYSTEM-PARAMETER	VOLATILITY CONNECTIVITY IS...TO VALUE IS VALUES ARE...THROUGH HAPPENS...TIMES-PER	9

System
Dynamics

PROCESS, EVENT
SET, ENTITY
INTERVAL
CONDITION
SYSTEM-PARA-
METER

VOLATILITY
VOLATILITY-SET
HAPPENS...TIMES-PER
VOLATILITY-MEMBER
TRIGGERED BY
INCEPTION CAUSES
TERMINATION CAUSES
TRUE/FALSE WHILE
BECOMING TRUE/FALSE CALLED

9

Figure 3.3

example. The following brief description, slightly modified, of a Medical Monitoring System (MMS) was taken from Stevens (Stevens 74):

A patient medical-monitoring system is required for a hospital. Each patient is monitored by an analog device which measures factors such as temperature, blood pressure, pulse rate, and so on. The program reads these factors on a periodic basis. For each patient, safe ranges for each factor are specified (e.g., patient X's valid temperature range might be 98 to 99.5 degrees F.). If a factor falls outside a patient's safe range, the nurses' station is notified.

Clearly, the above is incomplete as a specification for such a system, but it is adequate for illustration purposes.

First of all, note that the MMS is a real-time system - when it is running, it interacts on a continuous basis with the patients being monitored. Also, the system presumably maintains a database of "safe" values, one set of values for

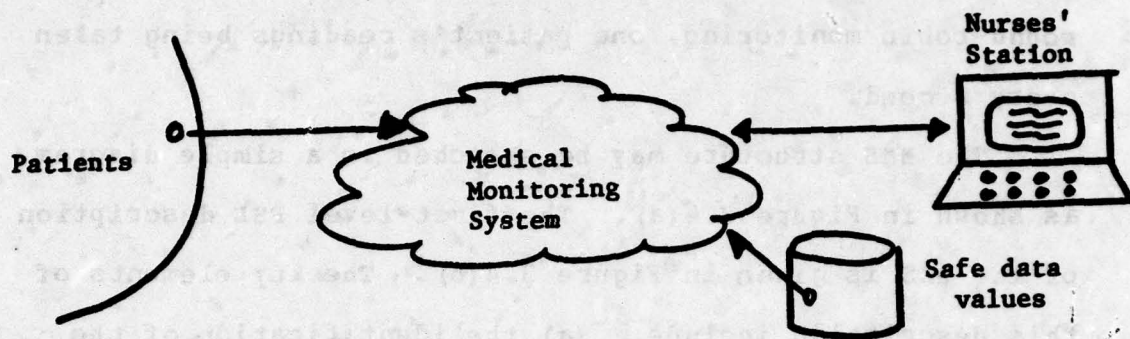
each patient. We will assume a maximum of 10 patients, and round-robin monitoring, one patient's readings being taken every second.

The MMS structure may be sketched in a simple diagram as shown in Figure 3.4(a). The first-level PSL description of the MMS is given in Figure 3.4(b). The key elements of this description include: (a) the identification of the INTERFACES that connect the MMS to the external environment; (b) the identification of the types of INPUT and OUTPUT data used or generated by MMS, and where obtained or used; (c) identification of a data SET, or collection of instances of data elements - the set of safe values; and, (d) the description of the PROCESS itself - the MMS analysis routines - and the inputs and outputs associated with it.

3.2.4 An Illustration of PSL - Part B.

We can expand and improve upon our earlier PSL description of the medical monitoring system by adding detail, describing more structure, etc. A more detailed depiction of the MMS might be as shown in Figure 3.5.

The MMS is shown there as consisting of three modules: one to monitor patient vital signs and compare to safe values, one to send an emergency message to the nurses' station in the event of monitored values exceeding the safe levels, and one to allow the nurses to modify the database of safe values.



(a)

Diagram for a Simple Medical Monitoring System

FIRST-LEVEL PSL DESCRIPTION FOR THE MMS

INTERFACE patients;
GENERATES patient-data;

INTERFACE nurses;
RECEIVES help-signal;

INPUT patient-data;
GENERATED BY patients;
RECEIVED BY monitor;
HAPPENS read-freq TIMES-PER minute;
USED BY monitor TO DERIVE help-signal;

OUTPUT help-signal;
GENERATED BY monitor;
RECEIVED BY nurses;
DERIVED BY monitor USING patient-data, safe-values;

SET safe-values;
RESPONSIBLE-INTERFACE nurses;
USED BY monitor TO DERIVE help-signal;

PROCESS monitor;
RECEIVES patient-data;
GENERATED help-signal;
USES safe-values;

INTERVAL minute;

DEFINE
read-freq SYSTEM-PARAMETER;

(b)

Figure 3.4

A new PSL description of the MMS, containing more detail than the first description, is given in Figure 3.6. The changes from the first description include: (a) the addition of the EVENT object types named alarm and noalarm, and the CONDITION object type named safe-values-exceeded; (b) the hierarchical decomposition of the MMS into three sub-processes; (c) the hierarchical decomposition of the INPUT, OUTPUT, and ENTITY data into components (GROUPS and ELEMENTS); and (d) the representation of the maintenance of the SET of safe values.

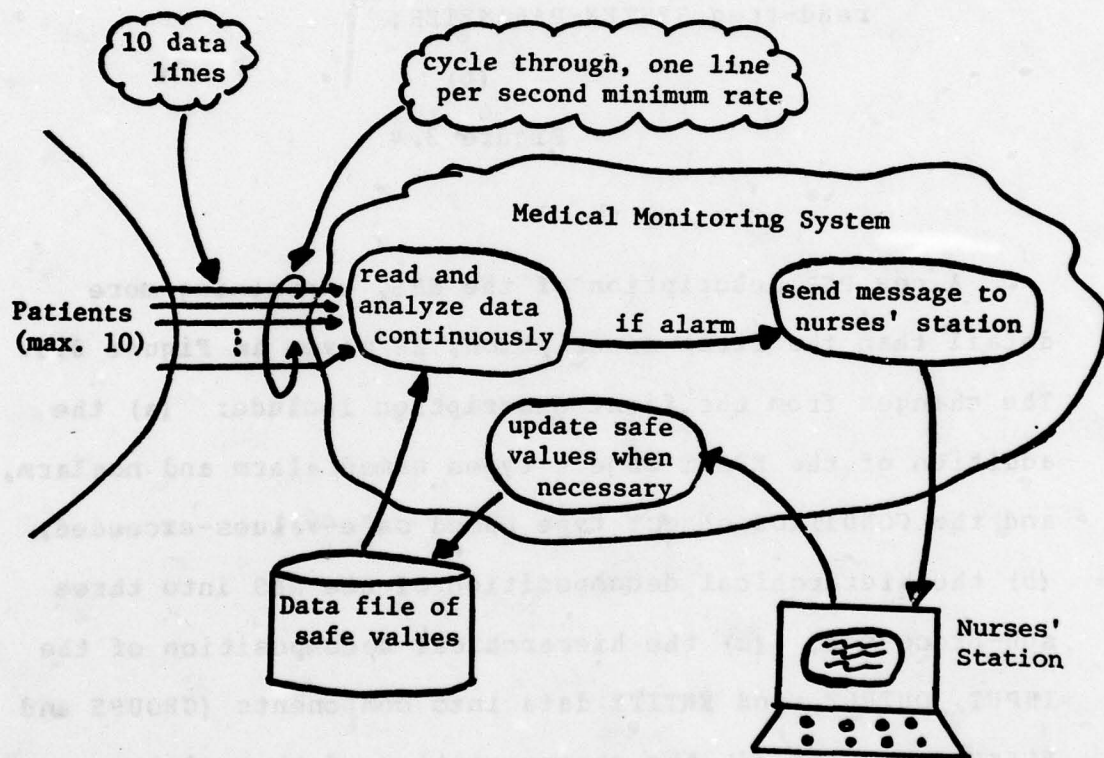


Figure 3.5

A More Detailed Description of the MMS

SECOND-LEVEL PSI DESCRIPTION OF THE MMS

INTERFACE patients;

GENERATES patient-data;

DESCRIPTION; between one and ten patients are
are to be monitored in real time.

Max. delay between readings on any patient
is to be ten seconds;

INTERFACE nurses;

GENERATES new-safe-values;

INPUT patient-data;

GENERATED BY patients;

RECEIVED BY analyze-vital-signs;

CONSISTS OF patient-id-number, patient-vital-signs;

HAPPENS read-freq TIMES-PER minute;

INPUT new-safe-values-info;

GENERATED BY nurses;

RECEIVED BY change-safe-values;

CONSISTS OF patient-id-info, new-safe-values;

USED BY change-safe-values TO UPDATE safe-values;

OUTPUT help-signal;

GENERATED BY send-alarm-to-nurses;

RECEIVED BY nurses;

CONSISTS OF patient-id-info, patient-vital-signs;

SET safe-values;

RESPONSIBLE-INTERFACE IS nurses;

USED BY analyze-vital-signs TO DERIVE help-signal;

UPDATED BY change-safe-values USING

new-safe-values-info;

CONSISTS OF safe-value-records;

CARDINALITY IS 10;

SECURITY IS authorized-nurses;

ENTITY safe-value-records;

CONTAINED IN safe-values;

CONSISTS OF patient-id-info, safe-values-info;

IDENTIFIED BY patient-id-number;

USED BY analyze-vital-signs;

UPDATED BY change-safe-values;

CARDINALITY IS 10;

VOLATILITY;

Any safe-value-records entity may be modified, deleted, or added to the data base by authorized nurses at any time. This will most commonly occur when the patient list changes;

GROUP patient-id-info;
 CONSISTS OF patient-name, patient-id-number;
 GROUP safe-values-info;
 CONSISTS OF safe-temp, safe-bloodp;
 GROUP patient-name;
 CONSISTS OF patient-first-name, patient-last-name;
 GROUP safe-temp;
 CONSISTS OF safe-temp-low, safe-temp-high;
 GROUP safe-bloodp;
 CONSISTS OF safe-bloodp-diastolic-low,
 safe-bloodp-diastolic-high, safe-bloodp-systolic-low,
 safe-bloodp-systolic-high;

ELEMENT patient-first-name, patient-last-name;
 ELEMENT patient-id-number;
 IDENTIFIES safe-value-records;
 ELEMENT safe-temp-low, safe-temp-high;
 ELEMENT safe-bloodp-diastolic-low,
 safe-bloodp-diastolic-high, safe-bloodp-systolic-low,
 safe-bloodp-systolic-high;

PROCESS monitor;
 SUBPARTS ARE analyze-vital-signs,
 send-alarm-to-nurses, change-safe-values;

PROCESS analyze-vital-signs;
 RECEIVES patient-data;
 USES safe-values;

PROCESS send-alarm-to-nurses;
 USES patient-data, safe-values;
 GENERATES help-signal;
 TRIGGERED BY alarm;

PROCESS change-safe-values;
 RECEIVES new-safe-values-info;
 USES new-safe-values-info TO UPDATE safe-values;
 PROCEDURE;

An authorized nurse updates the data base of safe values whenever an old patient is disconnected from the system, or a new patient connected. Also, current safe values may be altered as patients' conditions require;

EVENT alarm;
 WHEN safe-range-exceeded BECOMES TRUE;
 TRIGGERS send-alarm-to-nurses;

EVENT noalarm;
 WHEN safe-range-exceeded BECOMES FALSE;

AD-A074 911

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC METHODOLOGY FOR DESIGNING THE ARCHITECTURE OF COMP--ETC(U)
SEP 79 S L HUFF, S E MADNICK
CISR-P010-7906-12

N00039-78-G-0160

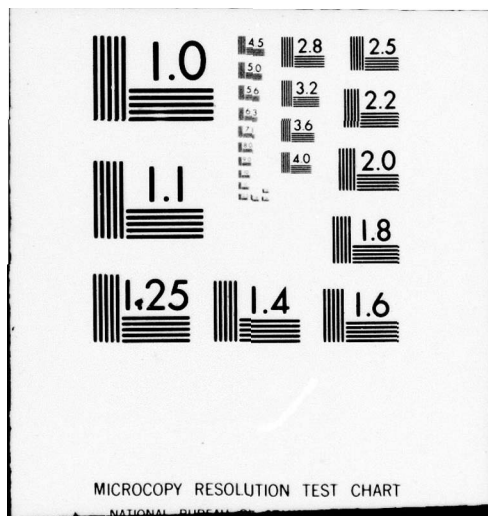
NL

UNCLASSIFIED

2 OF 7

ADA
074911






```

CONDITION safe-range-exceeded;
    BECOMING TRUE CALLED alarm;
    BECOMING FALSE CALLED noalarm;
    TRUE WHILE;
        Any vital sign for any patient connected
        to the MMS falls outside the defined safe
        range;
    FALSE WHILE;
        Vital signs within safe range;

INTERVAL minute;

DEFINE read-freq SYSTEM-PARAMETER;

```

Figure 3.6

3.2.5 PSL As a Design Tool.

An important issue which we address at this point is, to what extent does PSL serve as a designer's decision support system, i.e., play the role of a design aid? An important early motivation for the development of PSL/PSA was to help automate the task of system design, as opposed to documentation (Teichroew and Sayari 71). However, experience with PSL and other similar tools has seen them used primarily as documentation techniques, not design techniques (Teichroew and Hershey 77).

For example, in the MMS design illustrated in Figure 3.5 above, the designer (the author) decided, intuitively, that a three-module decomposition of the monitoring routine would be appropriate. The only justification for picking these three modules, with these functional charac-

teristics, was that they "seemed reasonable," based largely on previous general software design experience. Once the function of each module, and the data acted upon or interchanged among them, had been mentally worked out, PSL was then employed effectively to formally describe the scheme. However, PSL itself did not directly aid the designer in deciding on the system's structure, or on the functions of the components.

It is reasonable to infer, then, that specification (or "problem statement") languages like PSL, while they may be effective tools for gathering together and documenting information important to system design, in general do not fulfill the role of a methodology for guiding or assisting designers in conceiving system architectures. In contrast, the SDM approach is specifically oriented toward design assistance, and only secondarily towards documentation. This observation raises the question whether there might be a fortuitous combination of SDM and a scheme such as PSL/PSA that would effectively address both design support and documentation. Such a possibility is a subject for additional research, but is not addressed further in this thesis.

* * * * *

Research and practice in the field of requirements specification and preliminary system design has given rise to a number of important new concepts. However, it has also gen-

erated considerable confusion over interpretation of these concepts, much of which is due simply to a confusion over terminology. The purpose of the next section is to attempt to clarify some of the issues and problems in the requirement specification and system design fields. We will attempt to shed some light on certain heavily-used, but vaguely-defined terms, and to relate this present research - its objectives, methods, and concepts - to other work in the area, including application of PSL/PSA, in the context of a simple framework to be presented there.

3.3 TERMINOLOGY AND CONCEPTS: SOME CLARIFICATIONS.

The literature that focuses on the functional development phase of computer system design and development efforts exhibits much variation in content. There seem to be few unambiguous reference points - researchers, authors, and system designers have not yet agreed on precise terminology to describe what they do. This lack of agreement, understandably, adds confusion to both research and practice in this area.

In this section, we examine certain key terminological and conceptual topics, attempt to remove some of the ambiguity surrounding them, then consolidate them in a simple framework. This framework will prove useful in thinking about requirements specification and system development activities in general, and for relating the present research activity with other projects in this area.

3.3.1 Levels of Procedurality.

A widely used, but frequently misunderstood concept is that of procedurality. Programming languages are frequently described as being either "procedural" or "non-procedural" in nature. The usual definition centers on the distinction between stating or describing what is to be done (non-procedural) as opposed to how it is to be accomplished (i.e., the "procedure" to be followed, hence procedural). The main

problem with this distinction is that it is put forth as a black-versus-white characterization. In practice, no clear dividing line exists between the two alternatives; rather, they should be viewed as the ends of a continuum, thus:

procedural <-----> non-procedural

There are, then, different levels of procedurality - different degrees to which a statement (in particular, a requirement statement) exhibits either a "what to" or a "how to" nature. Furthermore, whether a particular statement, command, etc., is viewed as procedural or non-procedural depends on the viewpoint of the person involved. The following example should help clarify this distinction.

Consider an assembly language programmer, involved in writing the code for a calculation module which is to become the DCF (Discounted Cash Flow) subroutine for a financial analysis package. Faced with the task, say, of adding together two values, he would think in terms of instructions such as the following:

```
L    1,A    (Load "A" into register 1)
A    1,B    (Add "B" to contents of register 1)
ST   1,C    (Store results of addition in "C")
```


Now, conventionally, this set of instructions would be described as procedural in nature. The programmer has to know, for example, that the "procedure" for adding two numbers together involves loading a register with the first number, adding the second into the register, then storing the sum. Such considerations as loading and storing the registers, not to mention the use of assembler mnemonic codes and instruction format, are viewed as "how to," or procedural, aspects of performing this task.

In contrast, suppose the programmer were to make use of some high-level language, such as Fortran or PL/1. The addition operation might then be coded as (using PL/1)

$C = A + B;$

Now, from the assembly language programmer's point of view, such an encoding of the task is non-procedural: he no longer needs to be concerned about loading or storing registers, or about the other procedural aspects of the task. He can express the "what" aspect directly. (8)

This is the point at which the standard characterization of the distinction between procedural and non-procedural would end. However, suppose we go one step further,

(8) Of course, someone or something must worry about register loading, etc. In this case, the burden is shifted to the PL/1 compiler, or, if you prefer, the compiler designer.

and examine the same task from the point of view of, say, an eventual user of the financial analysis package (a non-programmer). As far as he is concerned, a typical non-procedural command might simply be

DCF PROJECT_X

which would execute the DCF module upon a given set of data, PROJECT_X. This command states, at his level of concern, what is to be done. The fact that, at some point within the DCF module, two values had to be added together using the PL/1 statement $C=A+B;$, is a procedural issue concerned with how the DCF calculation is to be carried out. So that which was viewed as non-procedural by the assembly language programmer, is clearly a procedural issue as far as the end user is concerned.

To summarize, then, the procedurality level of a formal language, command, etc., is not absolute, but rather must be related to the viewpoint of the user of that language, and the nature of the decision problem upon which he is working.

An important reason for presenting this detail here is that problem statement languages (e.g., PSL) are often characterized as being "non-procedural." The appropriateness of such a characterization depends very much on the user's point of view. If the user is a system documentor, such a

characterization may be appropriate; for a user who is trying to design a system, the characterization may be quite inappropriate. We will elaborate this distinction further at the end of this section.

3.3.2 Types of Requirements.

The literature in the systems design field also exhibits considerable ambiguity about what is meant by the "requirements" for a software system. Part of the reason for this is that the notion of requirements is very general. The term itself is used in many different contexts, and as a result these different contexts start becoming blurred in the mind of the reader, as well as designer. For example, systems analysts refer to "information requirements analysis" as well as "system requirements specification"; to "functional requirements" as well as "design requirements"; and to "user requirements" as well as "software requirements specification." There is a growing number of methodologies (discussed in more detail shortly) that purport to address the problem of "requirements specification," whatever that may be defined to be in any particular case. Examination of some of these methodologies indicates that, not infrequently, one person's "requirements" turn out to be another person's "detailed system design."

Now, at a high enough level, the concept of a system's requirements seems quite clear: these are statements of what the system is to do. Unfortunately, this definition is too general to be of much use. For example, "the system should help me control my inventory" is a requirement statement, as is "the file selection module must verify that file names are no longer than six characters." Clearly, these statements (a) are at different levels of abstraction, and (b) exhibit different degrees of procedurality.

In fact, it is insightful to classify requirements statements along each of these dimensions. This idea is pursued further in Section 3.4.4, in the context of the framework described there. One objective of that framework is to more precisely clarify the meaning of "requirements" especially with respect to the system development cycle.

3.3.3 Processes and Capabilities.

As system requirements move toward (a) lower levels of abstraction, and (b) higher degrees of procedurality, they are also altered along another key dimension: they are transformed from statements of capabilities (which the target system is to possess) into statements regarding processes.

The difference between capability-type requirement statements (CS's) and process-type statements (PS's) may be highlighted with some examples. Typical CS's (taken from a

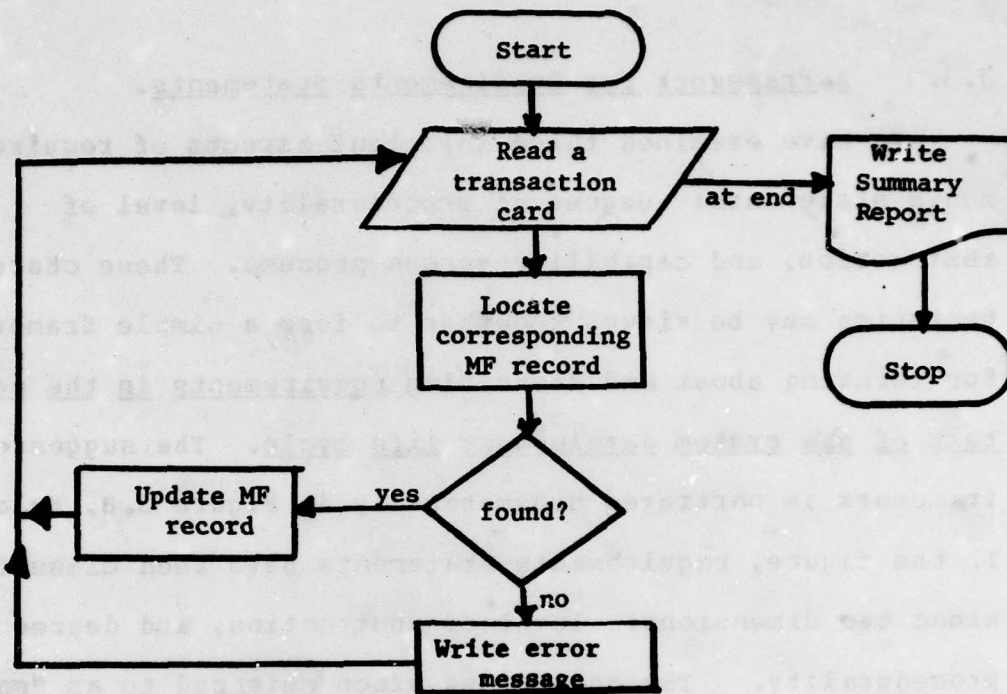
set of requirements for a data base management system, discussed in greater detail in the next section) might be:

"Inter-file relationships can be described at run time";

"The maximum size of a field is at least 100 characters";

"The system will have a report break control feature."

In contrast to these are process-oriented statements. A relatively detailed example of a PS is a common program flowchart, for example, Figure 3.7(a). Many other forms, at varying levels of detail, may also be found. Another example would be I.B.M.'s HIPC diagrams, and SofTech's SADT charts; both are graphical approaches to specifying PS's. The Requirements Statement Language (RSL), developed by the Ballistic Missile Defense Advanced Technology Group (Alford 77) and the Problem Statement Language (PSL) discussed earlier, are typical formal language mechanisms for stating process-oriented requirements. For comparison purposes, an example of the latter, duplicated from an earlier example, is shown in Figure 3.7(b).



(a)

INPUT patient-data;
 GENERATED BY patients;
 RECEIVED BY monitor;
 HAPPENS read-freq TIMES-PER minute;
 USED BY monitor TO DERIVE help-signal;

OUTPUT help-signal;
 GENERATED BY monitor;
 RECEIVED BY nurses;
 DERIVED BY monitor USING patient-data,
 safe-values;

PROCESS monitor;
 RECEIVES patient-data;
 GENERATES help-signal;
 USES safe-values;

(b)

Figure 3.7

Examples of Process-oriented Requirement Statements

3.3.4 A Framework for Requirements Statements.

We have examined three important aspects of requirements statements: degree of procedurality, level of abstraction, and capability-versus-process. These characteristics may be viewed together to form a simple framework for thinking about and describing requirements in the context of the system development life cycle. The suggested framework is portrayed schematically in Figure 3.8, below. In the figure, requirements statements have been classified along two dimensions: level of abstraction, and degree of procedurality. The activities often referred to as "management information requirements analysis" (Taggart 77) fall into the upper left quadrant of the diagram. Of course, not all systems development efforts are aimed at providing information to managers, but usually some user clientele is identifiable, and activities to elicit their needs, at a relatively high logical level, generally initiate the system development process.

As the process progresses, requirements are made more specific, often through some sort of hierarchical decomposition. Attempts are made to identify errors, omissions, conflicts, and other such problems with the requirements. Iterations are common. Often, problems encountered with "lower level" requirements necessitate alterations at "higher" (closer to the user) levels. In fact, it has been

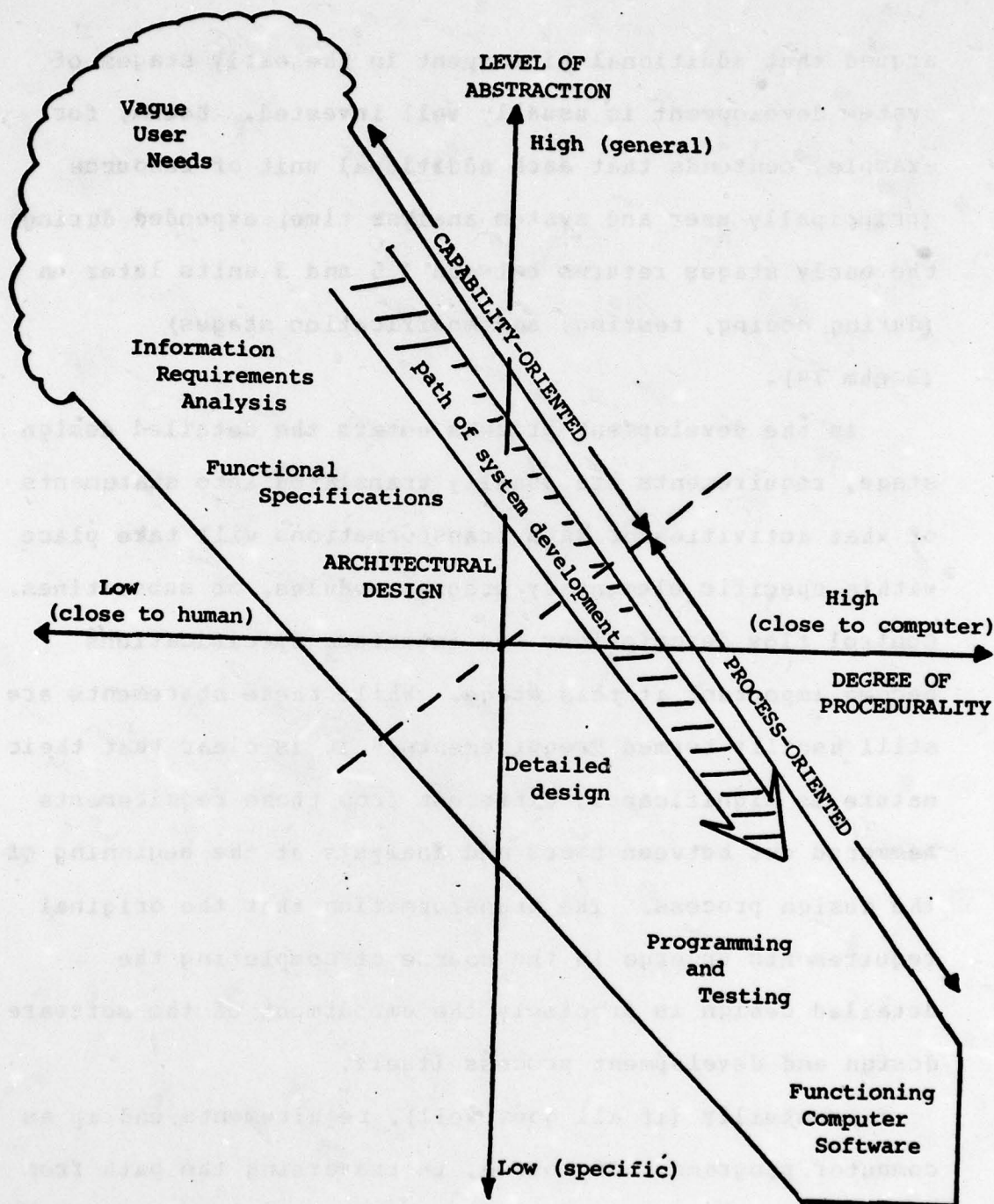


Figure 3.8

A Framework for Requirements Statements

argued that additional time spent in the early stages of system development is usually well invested. Boehm, for example, contends that each additional unit of resource (principally user and system analyst time) expended during the early stages returns between 1.5 and 3 units later on (during coding, testing, and modification stages) (Boehm 74).

As the development process enters the detailed design stage, requirements are usually translated into statements of what activities or data transformations will take place within specific elementary program modules, or subroutines. Control flow descriptions and interface specifications become important at this stage. While these statements are still usually termed "requirements," it is clear that their nature is significantly different from those requirements hammered out between users and analysts at the beginning of the design process. The transformation that the original requirements undergo in the course of completing the detailed design is precisely the embodiment of the software design and development process itself.

Eventually (if all goes well), requirements end up as computer programs. Of course, in traversing the path from user needs to PL/1 statements, considerable iteration is usually required, most often to clarify ambiguous or inconsistent requirements, or to fill in missing ones. It has

been observed (Bell 76) that, even under the most stringent conditions (e.g., designing the systems to support the Apollo moon launchings), it is effectively impossible to make an original set of requirements specifications complete. The necessity for iterations should not be viewed as an aberration of what ideally ought to be a linear activity; rather, it is an inherent, important part of the overall system design process.

A somewhat different way of characterizing the system development activities is in terms of capability statements and process statements. Initial conceptual work, groping by users and analysts as to the nature of the decisions requiring support, information needs, and so forth, usually lead to capability-type requirement statements ("the system must be capable of supporting up to five users simultaneously, "the system must support both on-line and batch access," etc.). As the design activity proceeds, process-type requirement statements usually emerge. These PS's serve to guide and document detailed design and programming activities to follow.

The transition from CS's to PS's to coded software is not (necessarily) very "neat," however. In many system development cases, the CS's are never formally stated at all; they occur by default, by designer "assumption" (which are frequently at odds with users' "assumptions"), or by

necessity (because other requirements constrain the design in various ways, often without explicit recognition). Also, it is not always all that clear whether a particular requirement statement is a CS or a PS. For example, the requirement that "the system should be able to print the output records in both sorted and unsorted form" specifies both a capability (being able to sort a file) and a process (sorting).

The distinction between CS's and PS's is important for the following reason: essentially all the currently available tools and techniques for aiding and guiding the system development activities are process-type techniques.

Included in this group are:

- HIPO (IBM)
- TAG (IBM)
- SOP (IBM)
- ADS (Honeywell, and ISDOS)
- SADT (SofTech Inc.)
- PSL/PSA (ISDOS)
- SREM and RSL (TRW Inc. and DMDATC)
- HOS (Draper Labs)
- HDM (SRI).

Consider again, for instance, the case of PSL. PSL is usually referred to as a non-procedural language for defining and describing a software system. However, keeping in mind that procedurality is really a continuum, a descriptive scheme such as PSL appears quite non-procedural as compared, say, to the assembly code eventually used to implement the system, but at the same time appears rather procedural as compared to the original English description of what the system was supposed to do. In particular, the PSL statements serve to structure an initial functional description of a system into (1) a set of data descriptions, (2) a set of processing descriptions, and (3) a set of other, miscellaneous items (EVENTS, CONDITIONS, INTERVALS, and the like). PSL, then, presents a relatively process-oriented description, consequently falls somewhere in the lower right quadrant of the diagram in Figure 3.8.

This is an important point, and warrants further elaboration. Conventionally, PSL is referred to as a language for stating software functional specifications. However, the point being made here, by way of example, is that

"if a factor falls outside the patient's safe range,
the nurses' station is to be notified"

is a good example of a true "functional" specification (one that is rather general in scope). However, the PSL statements

```
PROCESS send-alarm-to-nurses;  
  USES patient-data, safe-values;  
  GENERATES help-signal;  
  TRIGGERED BY alarm;
```

form part of the description as to how the previous function is to be carried out: a procedure (subroutine?) named "send-alarm-to-nurses" will be executed under certain conditions ("alarm"), will read the current data from the monitoring lines, will read the corresponding safe values from the patient's data file, and will send an appropriately formatted message to the terminal at the nurses' station. The PSL description is much more procedural and process-oriented than the earlier functional specification from which it was derived.

We can generalize the foregoing argument to essentially all well-known "functional specification stating" tools, languages, media, etc. References to these schemes, including the ones listed above, may be had through various reviews, such as (Cougar 73), (Teichroew 72), and (Burns 74). While the various techniques differ substantially in detail, the above comments regarding PSL apply in all the cases we have examined: that these schemes generally are positioned toward the process-oriented, procedural end of the spectrum presented in Figure 3.8; that they tend to be design documentation (or program documentation) techniques as opposed to design decision support systems; that they come into play

after the architectural design of the system has been created, not before or during and (this is the really confusing part) that they are frequently referred to, both by their creators, and sometimes even their users, as methodologies for functional specification, or problem statement, or system design.

The argument at this point is not that there is anything wrong with these PS methodologies per se, but that there are important steps in the system design process that must be taken - either planned or not, either supported or not - before process-oriented tools such as these can be brought into play. Central among these precedent activities is architectural design. Designers who seek to employ the process-oriented design aiding tools too soon in the development life cycle are, in effect, institutionalizing the tendency to underplay, skip over, or at the worst totally ignore that part of the functional development phase which addresses capability-type requirement specifications. Our research in requirements definition and software architectural design is centrally concerned with this problem.

3.4 EXPRESSING FUNCTIONAL REQUIREMENTS.

It would be useful to pause for a moment at this point to review some of the issues that have been raised, and points that have been made so far.

The focus of the SDM research is the problem of architectural design of software systems. The general approach developed by Andreu involves the application of partitioning algorithms to a graph representation of a design problem, and the interpretation of the resulting clusters and linkages as design sub-problems. Since in this chapter we are concerned especially with the problem of expressing functional requirements for a target system (to be used as "input" to the SDM procedures), and since there is a number of well-known "requirements specification" systems currently available, in Section 3.3 we examined a representative system: PSL/PSA, developed out of the ISDOS project. We then turned to an examination of certain terminological and philosophical issues that tend to cloud discussions of system design and software engineering. In the light of this examination, we argued that PSL, while possessing its own strengths and weaknesses, was in fact more oriented towards documentation than design, was more process-oriented than capability-oriented, and was more procedural than would be appropriate at the architectural design stage.

We concluded, then, that PSL, and by extension other methodologies exhibiting similar characteristics, are not very appropriate mechanisms for expressing functional requirements as a lead-in to system architectural design. An alternative approach for addressing this problem, which we call the "template approach," is described in this section.

3.4.1 The Format of Typical Functional Specifications.

The SDM methodology takes as input a set of functional requirement specifications for the target system, and as such is dependent on both the existence and the appropriateness of the specifications.

First of all, it is clearly necessary that a system's requirements be formally stated - i.e., written down - before SDM may be applied. Unfortunately, it is not at all uncommon for the requirements for proposed systems to never be committed to paper, especially in the case of smaller systems or systems being developed "in-house" (as opposed to contracted development). Nevertheless, for our purposes we will assume that this first step has been taken, that requirements have been generated in some written form.

Then, given that specifications have been formalized and written down, the second problem concerns the appropriateness of the format in which they have been stated. Three

important characteristics of requirement statements to be used in the SDM methodology include:

1. unifunctionality - each statement describes a single function (not multiple functions) to be featured in the target system;
2. implementation independence - each statement should be implementation free, i.e., ought to specify what is required of the target system but not how that requirement is to be met;
3. common conceptual level - all requirement statements should be, to the extent possible, at the same level of generality, or abstraction.(9)

In order to test out some of the earlier SDM concepts, Andreu managed to locate a set of requirement statements(10) for a database management system, that came "pre-packaged" in a format reasonably suitable for the analytical approach he had developed. As examples of these requirements, the following three are typical:

"Inter-file relationships can be described at run time;"

"The maximum number of interrelated files is at least ten;"

"User can cancel active request without loss of data."

(9) See (Andreu 78) for a more complete discussion of the characteristics of "good" requirement statements, in the SDM context.

(10) The requirements used by Andreu were issued by a U. S. Government agency as part of a procurement procedure.

The full list of Andreu's DBMS requirements is given in Appendix B.

Even though these requirements were in a roughly appropriate format from the outset, Andreu found that they had to be examined rather closely, and a number of them had to be edited somewhat, in order that they possess the three characteristics outlined above.

Unfortunately, functional requirements, when they formally exist at all, generally are not expressed in this fashion. More typical is the paragraph describing the requirements for the Medical Monitoring System, discussed earlier (see page 85). As another example of "typical" requirement statements, consider the following specification for the file system portion of an operating system specified by Honeywell for the U. S. Navy's All Application Digital Computer (AADC).

"The definitions of logical files are carried out by a collection of tasks directed towards file creation, file retention, file destruction, and file access actions. These tasks are accessible to other OS tasks and to application tasks. File definition tasks utilize the input/output tasks to manipulate and create various directory records (not PF, permanent file, directories) of files. Requests must be sent to the basic executive to initiate file definitions and to confirm access privileges to protected information. File definition tasks provide a user (system or application) with a mechanism for establishing logical files. The mapping function between logical and physical descriptors are established and the protection machinery invoked. Files will physically exist on devices such as drums or tapes. Since different physical devices usually exhibit differing physi-

cal capabilities, the logical file description will establish the logical capabilities of the file relative to the supplied device (Honeywell 72)."

The problem is, then, that if the SDM methodology for architectural design is to be widely applicable, it will be necessary to have a means of "translating" such typical functional specification statements into an appropriate form - i.e., a form exhibiting the three characteristics discussed earlier. An initial step towards defining such a mapping is proposed here.

3.4.2 Requirement Statement Templates.

We would like to have a simple procedure by which we could map general English-prose style requirement statements to a format suitable for input into the SDM procedure. As a first step in this direction, it would be useful to be able to identify a set of requirement statement types, or "templates," that might be used as a skeleton upon which specific sets of requirements could be constructed. Such a set of templates would

1. help to guide the thinking of the analyst in setting up the system specification for SDM analysis;
2. help to insure that the resulting statements met the appropriateness criteria outlined earlier; and
3. form the basis for further study and research of the requirements specification process generally.

To determine such a set of specification templates, the DBMS requirement set employed by Andreu was studied in detail. While the set included over 100 original requirements, close examination indicated that many of the statements exhibited similar patterns. For example, the three statements

"Variable-sized fields can be defined";
"Record-level lockout capability";
"Report break control feature supported"

all basically state specific features the target system is to possess.

Similarly, certain other patterns are discernible in the DBMS requirements. Specifically, upon thorough analysis, seven templates were distilled from the 100 different statements. These seven templates are given in Figure 3.9.

Specific terms used in the template descriptions in Figure 3.9 are defined below.

Objects. Objects are defined to be of two types: items, and activities. Examples of item objects are:

- (file) size
- (interactive query) facility
- (user) request.

Examples of activity objects include:

- (system) set-up

REQUIREMENT STATEMENT TEMPLATES

A. Existence.

There (can/will) be <modifier> <object>

B. Property.

<Mod> <object> (can/will) be <mod> <property>

C. Treatment.

<Mod> <object> (can/will) be <mod> <treatment>

D. Timing.

<Mod> <object> (can/will) <timing relationship>

<mod> <object>

E. Volume.

<Mod> <object> (can/will) be <order statement>

<index> <count>

F. Relationship (Subsetting).

<Mod> <object> (can/will) contain <mod> <object>

G. Relationship (Independence).

<Mod> <object> (can/will) be independent of <mod>

<object>

Figure 3.9

- (database) maintenance.

Modifiers. Modifiers are strings of English adjectives that serve to further describe the associated object. Examples of modifier strings are shown above, in parenthesis.

Properties. A property is a word that describes some particular feature of the associated object. Examples include

- self-documenting
- queryable
- distributed.

Treatments. Treatments are words that describe something that can be done to the associated object. Examples include

- saved
- sorted
- locked.

Timing Relationship. Pairs of activity objects may be temporally related via timing relationships. For example,

- occurs before
- triggers
- occurs during.

Order Statements. An order statement specifies an order relation (< , <= , = , >= , >) between an object and a measure (defined below). Typical order statements are

- less than
- no more than (i.e., less than or equal to)
- at least (i.e., greater than or equal to).

Measure. A measure consists of a parameter and a unit. The parameter may be either a constant or a variable, and the unit may be either a simple unit (e.g., hours, dollars) or a compound unit (e.g., man-months, or dollars per month).

Examples of measures include

- 2 hours
- 95 percent
- 11 man-months
- 120 characters per second.

Imperatives. Each template may occur in either of two imperative forms, distinguished by the use of either "can" or "will." The use of "can" indicates that the target system is to be capable of supporting the requirement being described, but that in any particular implementation that feature may or may not be so utilized. In contrast, the use of "will" indicates that the feature described in that requirement must be included as an imbedded part of the target system. Generally speaking, only one form ("can," or

"will") makes sense in any given requirement statement. For examples illustrating the differences between the two forms, consider the following property statements:

"data fields can be null," and

"null fields will be identifiable."

The first statement indicates that it is possible (although not necessary) for data fields to be set to a null value, whereas the second statement requires that null fields be identifiable (i.e., distinguishable from zero or any other valid value).

An example of each type of template is given in Figure 3.10. The full set of statements obtained by transforming Andreu's original DBMS requirements into template form is given in Appendix C.

EXAMPLES OF REQUIREMENT STATEMENT TEMPLATES

A. Existence.

There will be database-level security facilities.
modifiers object

B. Property.

System status will be queryable.
mod object property

C. Treatment.

Database can be initialized using system utility.
object treatment mod

D. Timing.

Schema validation will occur before database usage.
mod object timing mod object
relationship

E. Volume.

Maximum recovery time will be no more than 24 hours.
mod object order measure
statement

F. Relationship (Subsetting).

Record selectn criteria can contain boolean
mod object relnship mod
conditional expressions.
mod (cont.) object

G. Relationship (Independence).

Schema definition will be indep. of database usage.
mod object relnship mod object

Figure 3.10

3.4.3 Side Effects of Expressing Requirements in Template Form.

One of the interesting, and potentially important, results stemming from translating the DBMS requirements into template form is the fact that certain kinds of modifications had to be made to the original statements in order to perform the statement mapping. As a result, the "normalized" statements better met the appropriateness criteria for the SDH methodology, and the activity of normalizing the statements proved beneficial in reducing ambiguity and bringing about their clarification.

Four different effects were observed during the normalization process. First, and probably most important, by working the requirements statements into template form, one is forced to consider exactly what each statement means, and how it ought to be expressed in terms of the templates. With a little practice, statements or parts of statements that are ambiguous or unclear tend to stand out, as they tend to obstruct the transformation of the statement into a template form. As an example, statement 70 asserts that

"Application is transportable to/from Agency's existing systems."

In assessing which form this statement ought to be mapped into, a first step is to ask what the left-object is. In this case, "application" is the clear choice. However, the meaning of "application" is unclear: does it refer to application programs to be run on the target database system, or to some other application? Additional investigation showed that the reference was to application programs previously developed by the agency on an earlier DBMS, which they wished to be able to transfer later on, if necessary, to the new system for which the requirements had been issued. The statement is also a property-type statement of the "will" variety. The statement may be made clearer as follows:

"Application programs will be transportable to/from
mod object property mod

the agency's other DBMS's."
mod (cont.)

In this case, as for most of the DBMS requirement statements used by Andreu, the statement was able to be mapped into template form rather easily, but in so doing, sufficient thought had to be given to the statement's meaning that imbedded ambiguities and other possible difficulties (the meaning of "application," here) could be exposed and corrected.

The second observed effect was that of having to break up a requirement into two or more pieces in order to map it into template form. For instance, the original requirement 53 stated:

"Users can direct output to the system printer."

In fact, this requirement states two different things: there will be a "system printer capability" in the target system (this was not stated as a separate requirement elsewhere), and output may (optionally) be printed on the system printer. Therefore, this statement was split into an existence

statement,

"There will be a system printer,"
mod object

and a treatment statement,

"User output can be printed using system printer."
mod object treatment mod

One of the characteristics of good requirements statements, for the purposes of the SDM methodology, is that each statement capture a single functional requirement. Thus, the splitting of requirements, as demonstrated above, in order to map them onto templates, represents movement toward this goal.

A third issue brought to light by the requirements mapping activity concerns the necessity for inclusion of certain parts of some of the original statements. This is well illustrated by statement 77, which was originally

"Capability to support two or more concurrent queries in different stages of processing."

This statement was mapped into a volume template, to say in effect that the number of concurrent queries can be at least two. A question arises, however, over what is meant by the phrase "in different stages of processing," and why this is part of the requirement statement at all. It demands, for example, an understanding of what is meant by "stages of processing" in this context, an ambiguous notion at best. Also, it seems to have strong implementation overtones that would be desirable to avoid. What is really required is concurrent query processing capability (possibly with certain performance restrictions not mentioned here). One is lead, therefore, to either eliminate the last phrase from the specification, or else seek clarification from the user (in this case, the issuing agency). The final template form of this statement would then be

<u>"Number</u>	<u>of concurrent queries</u>	<u>can be</u>	<u>at least</u>	<u>2 units.</u>
object	mod		order	measure

As with the previous case, it might also be necessary to include an additional statement specifying the existence of a concurrent query capability.

The final type of issue raised by the statement mapping process concerns errors in the source statements. It is somewhat surprising, given the detailed examination these

requirements have received already, that there would be any obvious errors remaining. However, consider statement 62:

"Average system recovery time is 2 hours over a 30 day period."

In translating this statement using a volume template, the nature of the order statement ("is," here) was studied. Presumably, the intent of this requirement was that average recovery time be no more than 2 hours over a thirty-day period, not that it be exactly equal to 2 hours. The corrected statement was taken to be:

"Average 30-day recovery time will be no more than 2 hours."

modifier	object	order statement	measure
----------	--------	-----------------	---------

3.5 SUMMARY.

We began this chapter with a close look at a prototypical requirements statement language, PSL. We argued on the basis of this examination that PSL, and other similar languages are not appropriate tools to set up initial functional requirement statements for analysis within SDM. Basically, we pointed out that the use of these languages presupposes a system architecture in the designer's mind, if not formally documented. We discussed some of the ambiguous and loosely-used terms central to the requirements problem area, and in so doing were led to an explication of a framework for requirements statements. We then turned to the development of our own approach to the requirement statement expression problem, and put forth the template idea as a basis for guiding statement development. The underlying rationale for the template approach, as well as examples and side benefits of its use, were discussed.

In the next chapter we turn our attention to the problem of extending the SDM representational model so as to allow the system architect to express certain design-relevant information not presently representable in the current requirements model.

Chapter IV

EXTENSIONS AND IMPROVEMENTS TO THE SDM REPRESENTATIONAL MODEL.

4.1 INTRODUCTION.

Underlying the SDM approach is a technique for modelling the design problem by representing a system's functional requirements and their interdependencies as an undirected graph with unweighted (binary) links. This basic model and methodology have been applied to some experimental systems ((Andreu 78) (Holden 78)), and have been found to be an effective means of determining an initial design problem structure.

The purpose of this chapter is to examine the representational scheme used within the Systematic Design Methodology, and to suggest certain extensions to enhance the modelling power of this scheme. The SDM basic model, together with the extensions discussed in this report, will be referred to as the "extended model."

The extended model is applied to a simple design problem, featuring 22 requirements for the design of a database management system. This system was also studied earlier by Andreu (Andreu 78). Comparisons with Andreu's representation are drawn. In later chapters we will examine graph par-

0

tioning methods that might be used with the extended model, and measures to reflect the goodness of partitions derived from it.

4.2 OVERVIEW OF THE BASIC MODEL.

The design structuring methodology reported in (Andreu 78) forms a basis for the present work. At the core of this methodology is a simple model (the "basic model") used to represent general design structuring problems.

The basic model is an undirected graph: a set of nodes, together with a set of unweighted connecting links. Each functional requirement in the system specification is represented as a separate node. A link joining two nodes corresponds to an interdependency between these nodes. Ways in which both nodes and links are determined are discussed below.

4.2.1 Generation of Nodes in the Basic Model.

Each node represents a single functional requirement of the target system. Desirable properties of these functional requirement statements include:

1. unifunctionality - each statement describes a single function (not multiple functions) to be featured in the target system;
2. implementation independence - each statement should be implementation free, i.e., ought to specify what is required of the target system but not how that requirement is to be met.;
3. common conceptual level - all requirement statements should be, to the extent possible, at the same level of generality, or abstraction.

In Andreu's research, the problem of creating the functional requirements for a system was not directly addressed; instead, they were taken as given.(11) One scheme for mapping English-language prose requirement specifications into an appropriate set of functional requirement statements according to the above guidelines was discussed in the previous chapter.

For the purposes of this report, the assumption will again be made that appropriate functional requirement statements are given to the system designer.

4.2.2 Generation of Links in the Basic Model.

Interdependencies between pairs of requirements are represented as links, or "edges," in the basic graph model. Andreu and others have discussed in some detail the interpretation of design interdependencies. For example, Andreu (Andreu 78, page 70) writes:

"In essence, two requirements are interdependent when one can think of plausible implementation schemes in which the two ought to be considered simultaneously for design purposes, the reason being that if such an implementation scheme was to be adopted, meeting these requirements would be one of the central considerations that the designer should take into account to tailor the scheme to the characteristics of the design in progress."

(11) Andreu did discuss guidelines for inspecting and verifying the requirement statements.

Furthermore, interdependencies fall naturally into two groups, termed concordant and discordant.(12) A concordant interdependency exists between requirements A and B if the implementation of requirement A would tend to simplify, assist, or otherwise make easier the implementation of requirement B. In contrast, a discordant interdependency exists between the two requirements if implementation of A would hamper, jeopardize, or otherwise make more difficult the implementation of B.

The basic approach suggested by Andreu for actually determining the interdependencies has the designer consider each requirement pair in turn, and mentally consider the alternative approaches he might follow in implementing the two requirements. The various implementation schemes so conceived are termed "mental models" of implementation. If the designer perceives a significant degree of interaction, in the context of his mental models, between a given requirement pair, he interprets the requirement pair as being interdependent.

The actual determination of interdependencies and their nature (concordant or discordant) is heavily designer-dependent. There is no intent within the methodology to remove or de-emphasize the designer's experience or judgment from -----

(12) These terms were suggested, in an unexpected surge of creativity, by Prof. J. Meldman. Andreu had earlier used the terms "tradeoff" and "concurrency."

the design activity. Rather, the methodology attempts to provide structure and simplification to the decisions the system designer must make. By only having to consider a pair of requirements at one time, rather than the entire set, the cognitive demands on the designer are greatly reduced. There is a price to be paid for this simplification, however: now the designer has easier decisions to make (pairwise comparisons), but more of them. While not as formidable a task as might appear at first glance, the complete assessment of interdependencies for a non-trivial problem involves considerable effort.

4.2.3 An Example.

To illustrate more concretely the ideas discussed in this paper, a specific real (but small) design problem will be studied in terms of both the basic model and the extended model. The problem concerns the design of a database management system (DBMS). A set of 22 functional requirements is assumed given (refer to Section 4.4.1 for a listing of these requirements).

Requirement interdependencies were assessed for this requirement set in the fashion described in the previous section. A total of 38 interdependencies were determined. Descriptions of each interdependency are given in Section 4.4.2 of this report, where the example system is examined in more detail.

These requirements and interdependencies may be displayed as a graph, following the basic model, as shown in Figure 4.1. Of course, determining the graph structure for the system requirements is only the first step in the basic design methodology. Further steps, including partitioning the graph appropriately, and interpreting design subproblems and their interactions, would normally follow. As this chapter is only concerned with representational issues, further steps such as these will be analyzed in later chapters.

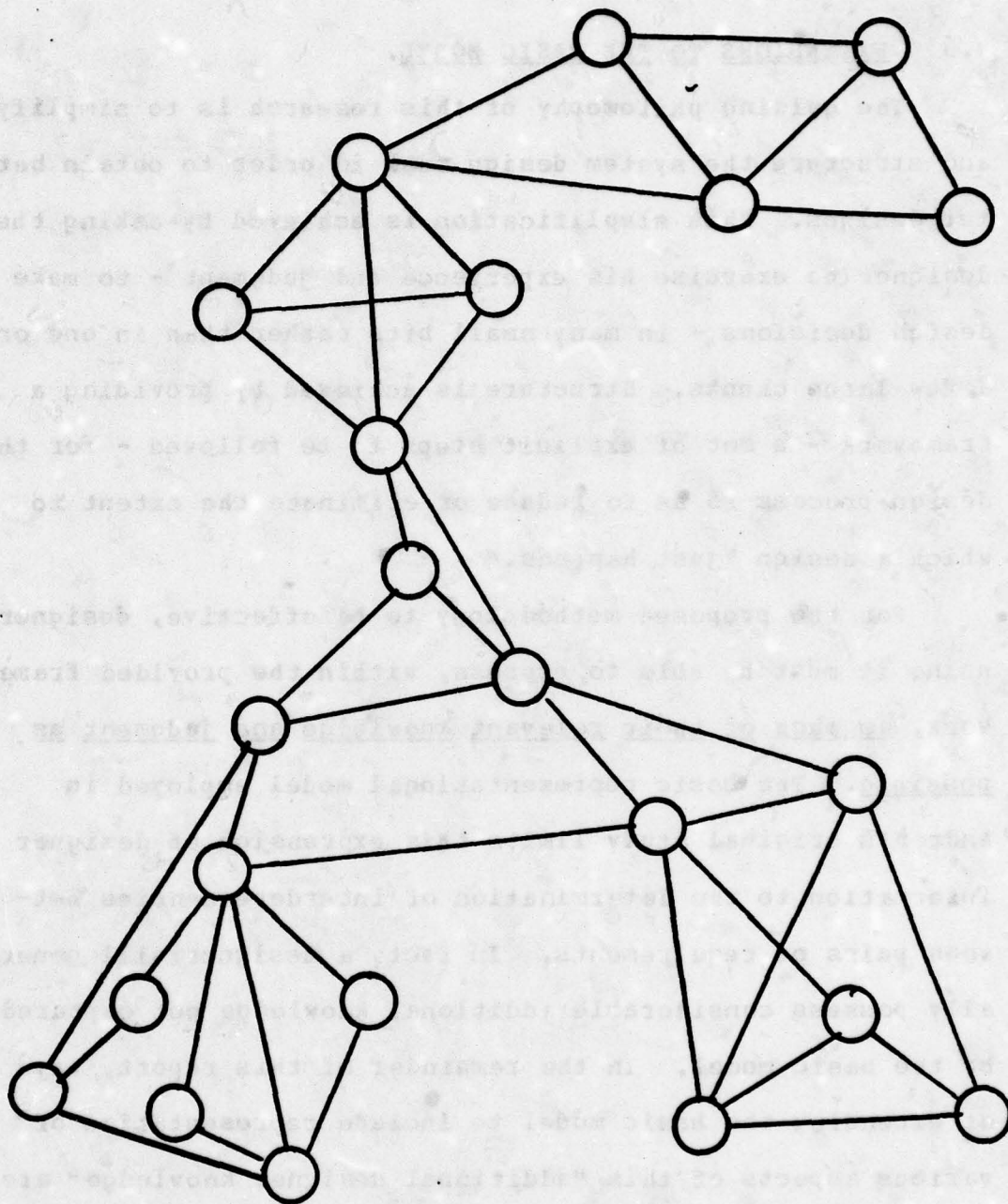


Figure 4.1

Graph representation of the 22-node DBMS design problem

140

4.3 EXTENSIONS TO THE BASIC MODEL.

The guiding philosophy of this research is to simplify and structure the system design task in order to obtain better designs. This simplification is achieved by asking the designer to exercise his experience and judgment - to make design decisions - in many small bits rather than in one or a few large chunks. Structure is achieved by providing a framework - a set of explicit steps to be followed - for the design process so as to reduce or eliminate the extent to which a design "just happens."

For the proposed methodology to be effective, designers using it must be able to express, within the provided framework, as much of their relevant knowledge and judgment as possible. The basic representational model employed in Andreu's original study limits this expression of designer information to the determination of interdependencies between pairs of requirements. In fact, a designer will generally possess considerable additional knowledge not captured by the basic model. In the remainder of this report, ways of extending the basic model to include representation of various aspects of this "additional designer knowledge" are presented and discussed.

It should be made clear that the intention of this report is to lay out and analyze various possible kinds of extensions that could be made to the basic model. No

choices will be made at this time as to which, if any, of these extensions will in fact be adopted for further research in this problem area. However, in conjunction with the example discussed in Section 4.4, the ease of application of each extension is examined briefly, and tentative conclusions about priorities regarding extensions to the basic model are drawn. These conclusions are further examined in light of the experience gained in a full-scale SDM application study reported in Chapter 7.

4.3.1 Interdependency Weights.

In the basic model, an interdependency either exists or it doesn't - there is no middle ground, no notion of the "strength" of an interdependency. Links within the graph representation of a design problem are binary in nature: the adjacency matrix is a matrix of ones and zeros.

There is nothing inherent in the design problem representation that necessitates binary links, other than a desire to work with as simple and parsimonious a model as possible. On the other hand, there is good reason to relax this requirement, namely, that important aspects of designer knowledge might thereby be included in the design problem model.

One possible extension of the basic model would be the association of a "weight" $W(i,j)$ with each link. The inclu-

sion of such link weights in the graph model can be likened to course grading: binary links would correspond to pass-fail grading, whereas weighted links correspond to standard (letter or numerical) grading. Just as standard grading allows an instructor to express more information about his or her students, so weighted links would serve to capture more of the designer's judgment concerning the relationships among system requirements.

There is a variety of possible ways in which such a weight could be defined and justified. For example, the weight $W(i,j)$ could be taken to represent the "closeness," or "strength of interdependence" of the requirements i and j in the context of the interdependency represented by link $L(i,j)$. With this interpretation, two requirements that are seen to be closely related, in implementation terms, would be connected by a link with a relatively high assigned weight.

Alternatively, link weights could be defined in a subjective probability sense. In this case, the weight $W(i,j)$ would represent the degree of uncertainty in the designer's mind that the requirements i and j will interact in implementation. This interpretation would be consonant with the uncertainty inherent in the interdependency assessment process itself. With this definition, a designer who is quite certain that two requirements i and j would be coupled in

implementation would assign a relatively high weight (close to 1) to the link $L(i,j)$; whereas if the designer believes that there is only a fairly small likelihood of the requirements being coupled, the assigned weight would be lower.

These alternative definitions of link weight give rise

		Subjective probability	
		low	high
Strength of Interaction	low	A	B
	high	C	D

Figure 4.2

Logical combinations of link weight interpretations.

to four logical combinations, as displayed in Figure 4.2.

In fact, these definitions are not completely "orthogonal." If, for example, two requirements are seen to be strongly related, according to the first definition, then there will be a tendency for designers to view the probability of their being related as high. Thus, it is reasonable to assume that most designers' weight assignments would fall in cells A and D in the above diagram.

4.3.1.1 Scaling Problems.

A decision to include link weights in the graph model gives rise to certain scaling problems. First, a range must be specified over which weights will be allowed to vary. While not absolutely necessary, compatibility with the basic model, among other reasons, suggests that link weights be chosen from the numerical range $[0,1]$.

A choice must also be made between requiring the designer to choose from "pre-set" weight values (e.g., low, medium, high) versus a continuous range of values. If pre-set weights are used, some mapping to numerical values must also be chosen (e.g., "low" corresponds to a numerical weight of 0.2, and so forth). All link weights must eventually be encoded numerically, for use in the graph partitioning algorithms.

Also, it may prove desirable to assign special meaning to certain weight values. For instance, a special "very high" weight category might be defined to allow a designer to express his conviction that two requirements absolutely must be included in the same sub-problem.

In contrast, a "very low" weight may be used to specify that two requirements must be in different sub-problems. However, there are also certain arguments for avoiding such deterministic weight assignments, in that they reduce design flexibility and partially preempt the design structuring methodology itself.

For the purposes of implementing extensions to the basic model, hard choices need not always be made among alternative representation issues ahead of time. A software package that designers would use to apply these architectural design methods could allow the user to select, from various options, those alternatives that most appealed to him, that he found easiest to use, etc.

Our experiences to date indicate that system designers most likely would not require the capability of specifying weights directly in numerical terms. The following scheme, using a simple three-way breakdown, has proven effective and easy to use:

Designer's Judgment About Weight	Code	Numerical Value
Strong	S	0.8
Average	A	0.5
Weak	W	0.2

This particular encoding of weights achieves a number of useful results. First, the three basic weightings may be easily interpolated, as shown below:

Weight:	S+	S	S-	A+	A	A-	W+	W	W-
Numerical value:	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

Secondly, all weights fall in the $[0,1]$ range, a fact useful for normalization and later decomposition purposes. Finally, the end points in the range (the values 0 and 1) may be reserved for "must be included" and "must not be included" categories, if so desired.

There is another, more subtle, set of scaling issues that involves the correctness, or accuracy, of link weight assignments. If, in assessing a set of requirements, a designer assigns a weight of, say, 0.85 to a certain link, how does he know that it should be 0.85, and not 0.84 or 0.86? Of course, he doesn't know for sure; he is simply trying to quantify as best he can what is inherently a ill-defined issue. This accuracy problem did not first arise with the introduction of link weights into the model. It was present in the basic model as well, in the guise of deciding whether particular links should or should not be included at all.

The original motivation for augmenting the basic model through the inclusion of link weights (as well as other extensions to be discussed shortly) is that designers could provide additional information, relevant to the design structuring problem, that is not captured by simple binary links. The fact that this information is not one hundred percent "accurate" should not prevent it from being

used. (13)

4.3.2 Information Linking Implementation Issues.

In the basic model, two nodes (requirements) are joined by a link when they are deemed to be "implementation interdependent" by the designer. In essence, links represent implementation considerations. In this view of requirements and implementation considerations, the focus is upon the nodes, with links being a kind of implementation "glue" which binds the nodes together.

A different way of viewing links is as "things," or logical entities, in their own right, rather than just bindings. To some extent, the appropriateness of viewing an implementation issue as a logical entity depends on the specificity of the statement describing that issue. Not all interdependencies can be specified by a designer with the same level of precision. In some cases, it is possible for designers to indicate in some detail the nature of the interaction that he believes will occur in the process of implementing a given pair of requirements. Requirements for "fast direct-access retrieval response" and for "ability to perform sequential file processing efficiently" may be

(13) There is a strong similarity between this argument and that put forth by Bayesian statisticians in defending the use of personal, or subjective, probabilities in statistical models.

assessed as concordant requirements since, the designer may believe, the use of the Indexed Sequential Access Method (ISAM) file organization is required in both cases. In such a situation, the link joining this pair of requirements represents more than just general implementation interdependence; in particular, it represents the use of ISAM file organization, a rather specific implementation scheme.

In other cases, a designer will not be able (at this stage in the design process) to be so specific. He may believe that a pair of requirements will prove to be discordant (interfere with each other) as the design proceeds, but may not, as yet, be able to specify how this interference will come about.

It is useful to treat interdependencies as separately identifiable issues for another reason also, namely, for documenting project design decisions. It often happens in design and development projects (especially large projects) that early design decisions are made by one person or group, and left undocumented. The responsible individuals may leave the project, or otherwise become "disassociated" from that aspect of the system. Later designers and implementors often come to question what the motivation might have been underlying earlier decisions. Having a well-documented "track" for each design decision would be quite valuable for determining how later design issues or system modifications ought to be implemented.

Now, viewing links as representing logical entities leads to certain questions about such entities: can these entities themselves be related in ways meaningful to designers and to the design structuring problem? The answer is yes, and the incorporation of such link relationships into the basic model provides additional useful extensions.

There are two different kinds of relationships between implementation issues discussed in this report. In this section, similarity relationships are addressed; implication relationships are addressed later.

4.3.2.1 Similarity Links Among Implementation Issues.

Two or more links may represent the same, or closely related, implementation issues. A simple example of this possibility is illustrated in Figure 4.3. The links joining requirements 1 and 2, and requirements 2 and 3, both represent the interdependency "ISAM organization," an implementation consideration through which both requirement pairs (1,2) and (2,3) are deemed by the designer to be interdependent.

In this example, the two links represent entirely the same implementation issue. In general, however, the degree of "sameness" between two or more implementation issues will usually be less than 100 percent in the eyes of the designer, due to inherent fuzziness in the specification of both functional requirements and implementation schemes. The

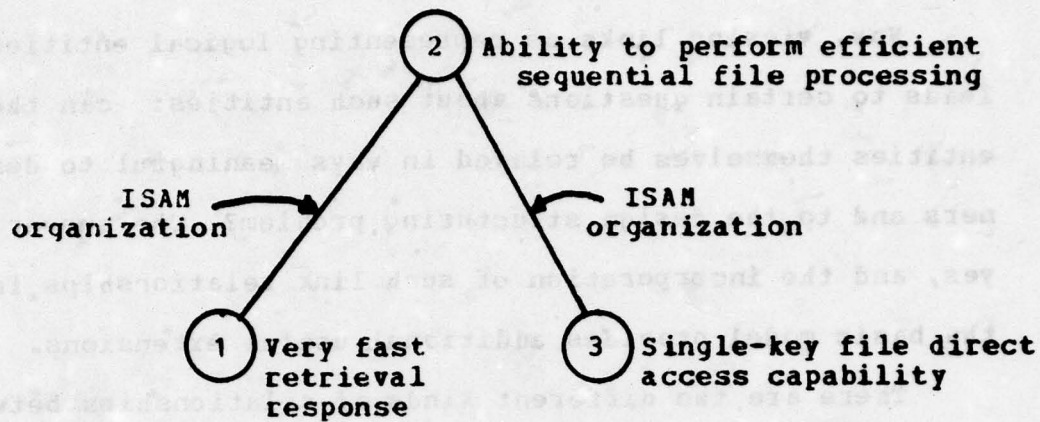


Figure 4.3

Example of Implementation Similarity Relationship

judgment as to whether a given pair of links "really" represent the same implementation issue is, again, a designer decision.

Going one step further, a weight factor could be associated with the similarity assessment to represent the extent to which the designer judges the two implementation issues to be the same. That is, such a weight would correspond to the extent of overlap between the implementation issues, in the designer's estimation.

4.3.2.2 Graph Representation of Implementation Issue Commonality.

Probably the easiest approach to the schematic representation question is to avoid it, by not actually incorporating implementation similarity information into the graph at all. Rather, the associated information could be kept in a table or matrix, to be included in the model but not actually depicted schematically.

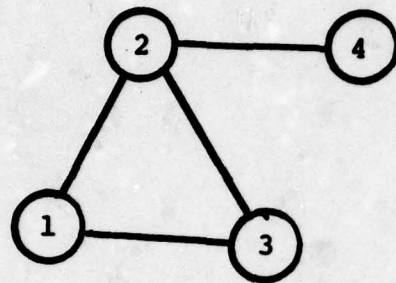
An alternative approach, which would lend itself to easy display, would be to define a new type of node - an "implementation node." This definition would extend the diagrammatic power of the graph representation. It would also provide a simple mechanism for representing various kinds of relationships between implementation schemes: as links between implementation nodes. Such links would, of course, be different in kind from links between requirement nodes; the latter would represent interdependencies between pairs of requirements, the former would represent relationships between the implementation schemes themselves.

As a diagrammatic technique, requirement nodes (R-nodes, henceforth) ought to be distinguished from implementation nodes (I-nodes). One simple approach would be to use circles for the former (as in the basic model) and, say, squares for I-nodes. Similarity rela-

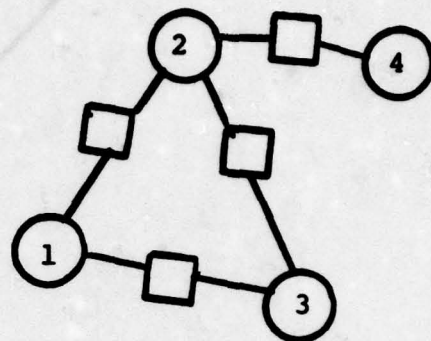
tionships between implementation issues may then be represented by undirected links between the corresponding I-nodes. Furthermore, the degree of similarity could be represented as a link weight, similar to the "strength of interdependence" weights discussed in Section 4.2.

These diagrammatic ideas are illustrated in Figure 4.4. Figure 4.4(a) shows a simple four-node graph using the style of the basic model. In Figure 4.4(b), implementation nodes are formally indicated and labelled (the specific implementation concept that each such node is intended to represent would presumably be described in accompanying documentation). In Figure 4.4(c), the fact that certain of the I-nodes are judged to represent similar implementation schemes is depicted using links connecting those I-nodes. Finally, in Figure 4.4(d), weights are attached to these links to describe the extent of overlap, or similarity, between the implementation issues, as perceived by the designer.

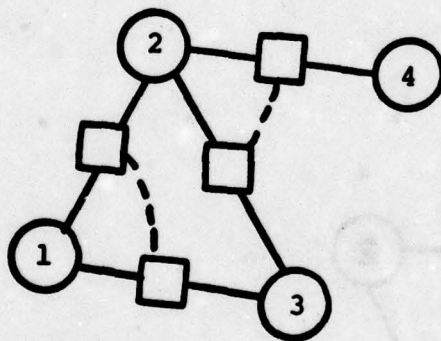
Viewing implementation issues as network nodes makes a new kind of interdependency - a multi-requirement interdependency - representable. For instance, requirements 1, 2, and 3 may all be interdependent, as a group, via some implementation issue X. That is, a single implementation issue may interrelate all three requirements at once, as illustrated in figure 4.5(a). This



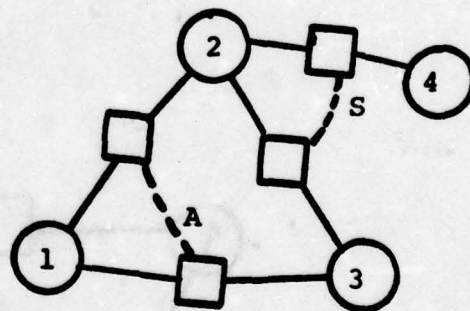
(a)



(b)



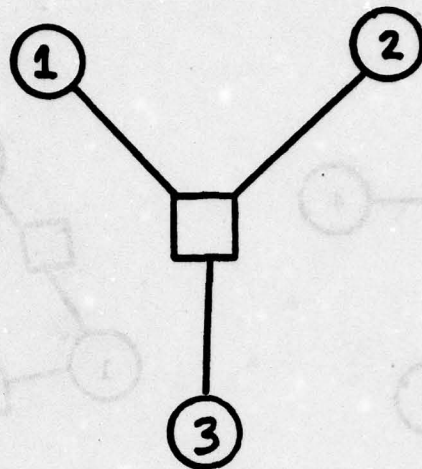
(c)



(d)

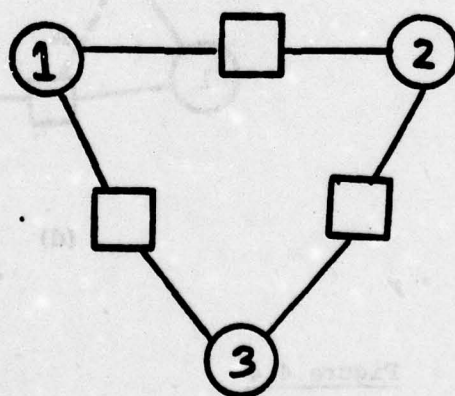
Figure 4.4

Requirement nodes and Interdependency Similarities



(a)

A multi-requirement interdependency



(b)

A trio of similar pairwise interdependencies

Figure 4.5

kind of relationship would be different from a trio of similar pairwise interdependencies (see Figure 4.5(b)), as it would more strongly suggest the appropriateness of clustering the three requirements into a common design sub-problem.

Additional kinds of information that may be represented using relationships between implementation nodes are discussed in the following section.

4.3.3 Representation of Implication Information.

It may prove useful in some cases to include still more kinds of semantic information regarding relationships among the various requirements and implementation schemes. Additional considerations not captured in the model extensions discussed so far include:

1. logical implications between requirements at the same level of abstraction, and between requirements at different levels of abstraction;
2. logical implications between implementation concepts;
3. logical relationships between requirements and implementation schemes.

4.3.3.1 Logical Implications Between Requirements.

It may be the case that a designer possesses information concerning relationships that exist between pairs of

requirements, even with no consideration being given to implementation issues. Perhaps the most common type of such a "functional" relationship (as compared with relationships derived from implementation considerations) is implication: "requirement 1 implies requirement 2"; or, stated somewhat differently, "if requirement 1 is to be included in the system specification, then it follows that requirement 2 must be included also."

It is important to note that such a functional relationship, as defined here, may exist independently of how requirements 1 and 2 are to be implemented. The logic of the relationship is contained in the semantics of the functional requirement statements alone, without consideration being given to implementation alternatives.

Lateral Implications.

Implication relationships may be further categorized as being either "lateral" or "hierarchical." The difference between these two types depends on the relative level of abstraction⁽¹⁴⁾ of the two requirements, as seen by the designer. Implication relationships between requirements that exist at the same level of abstraction are laterally

(14) The concept "level of abstraction" is widely used in the system specification and system analysis literature. It is, however, rarely defined, and is generally taken as a kind of primitive concept.

related, while those between requirements at different level are hierarchically related.

To make the above ideas clearer, some examples are developed below.

Examples.

Example As an example of logical implications between requirements, consider the pair of requirements

- "capability for collecting resource usage statistics for each submitted job," and
- "able to charge users by department and group for resources used."

These functional requirements are seen to be logically related even without considering implementation alternatives: fulfilling the second requirement implies fulfilling the first. Unless a system is able to account for resource usage, it will have no rational mechanism for meeting the second requirement. Furthermore, the designer would probably view these requirements as existing at a common level of abstraction - i.e., neither is logically contained within the other.

In passing, it might be noted that these requirements are also interrelated at the implementation level, since the implementation of the resource statistics collection subsys-

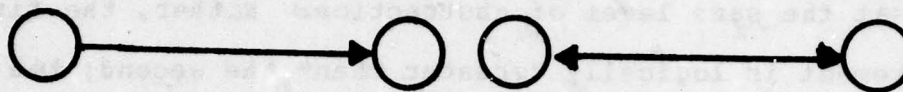
tem will undoubtedly be affected by considerations of the lines along which charges are to be collected and distributed. It is conceivable, although unlikely, that two requirements could be related through logical implication at the functional level, yet not be related at the implementation level.

As well as one-way implication, each requirement of a pair could logically imply the other. A possible example of such a pair might be the requirements

- "system must be accessible in conversational mode,"
- and
- "system must be convenient to use."

The designer may feel that an on-line system should always be more convenient to use than a batch system; contrariwise, in order for a system to be truly more convenient to use, it must provide an interactive user interface - i.e., it must be an on-line system.

Diagrammatically, these logical implications between requirement nodes could be represented with a conventional directed link, thus:



(a)
One-way

(b)
Bi-directional

Figure 4.6

Representation of Logical Implication between Requirement Nodes.

An arrowhead is used to distinguish implication relationships from implementation-level relationships.

Hierarchical Implication.

Requirements may be related in an implication sense and also exist at different (hierarchical) abstraction levels.

For example, a requirement A may be logically part of another requirement B.

Consider, for instance, the pair of requirements:

- "capability for report formatting," and
- "report formatting optionally automatic."

These requirements are arguably related in a functional sense, but not in the earlier sense of one implying the other at the same level of abstraction. Rather, the first requirement is logically "greater than" the second; the second does not make sense without the first.

It is not necessarily easy to distinguish hierarchical implications from lateral implications. Nevertheless, the differences do seem to be distinguishable in some cases. Since these two types of relationships may be seen to carry different kinds and amounts of design structuring information, it is appropriate to make a differentiation for the purposes of this report.

Hierarchical implication relationships could be represented schematically as shown below, the double line being used to distinguish from both lateral implications and from implementation relationships.

Requirement 1
hierarchically implies
requirement 2.

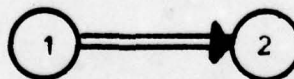


Figure 4.7

Representation of Hierarchical Implication Relationships.

4.3.3.2 Logical Implications Between Implementation Issues.

An argument has been made for the representation of similarity information joining pairs of implementation nodes (I-nodes). Other kinds of relationships between I-nodes may also be represented in the extended model.

The previous arguments regarding logical implication between pairs of I-nodes may be carried across to implementation nodes also. That is, the assumed existence of some implementation scheme X may be viewed by the designer as implying the inclusion of some other scheme, Y.

While it may even be possible to extend the distinction between lateral and hierarchical relationship types to this case, for practical purposes the designer generally does not have a clear enough picture of ultimate detailed implementation techniques during architectural design activity to make such relatively fine distinctions. Therefore, only one "general-purpose" type of logical implication relationship between I-nodes is considered for inclusion in an extended graph model.

As an example of such a relationship between I-nodes, consider the following three requirements:

1. "Very rapid record retrieval";
2. "Ability to produce sequential listing efficiently";

3. "Ability to access records based on value of key field."

Now, a designer might assess requirements 2 and 3 as being related via an implementation issue X: "ISAM file organization." Similarly, 1 and 3 may be seen to be related via Y: "In-core index tables." Finally, for the sake of illustration, the designer may believe that implementation issue X logically implies implementation issue Y; that is, the use of indexed sequential file organization as an implementation technique implies the use of in-core tables for record lookup.

Borrowing again from the earlier arguments regarding implications among requirements, a straightforward way of portraying I-node implications would be to make use of a directed link joining two implementation nodes. The example discussed above is so illustrated in Figure 4.8.

4.3.3.3 Logical Implications between R-nodes and I-nodes.

In the previous section, arguments have been given for including in the extended model various means for representing logical implication relationships between R-node pairs, and between I-node pairs. For completeness' sake, consideration should also be given to implications between an R-node and an I-node.

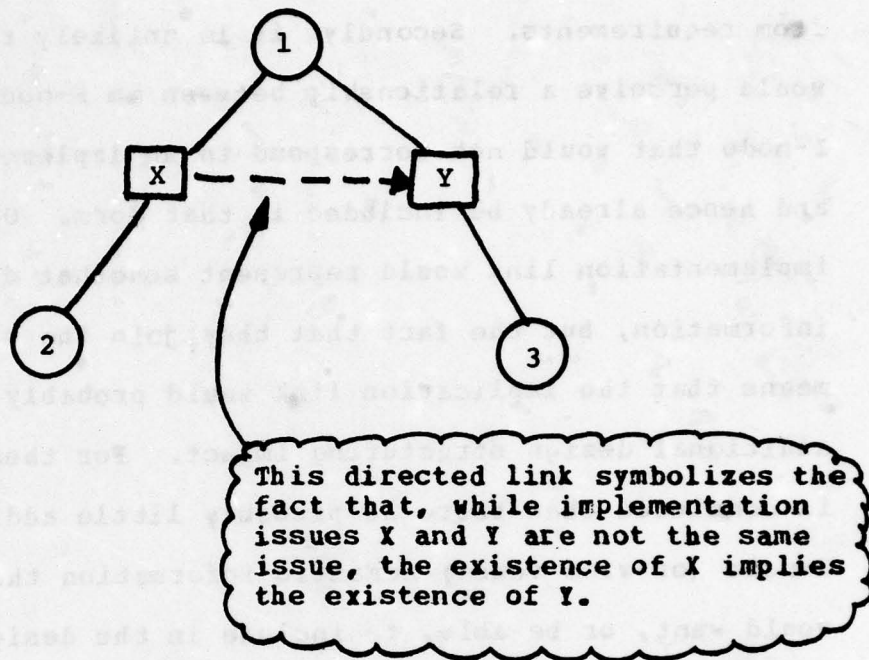


Figure 4.8

Representation of Implication Relationships between Implementation Nodes.

It is possible, in theory, to differentiate implication semantics from the conventional implementation coupling already included in the model between R-nodes and I-nodes. Nevertheless, it is unlikely that there would be a real need for a capability to represent such implication information, for two reasons. First, presumably all implications would have the same "orientation": from R-node to I-node. After all, the requirement nodes are the starting point for the whole process, and implementation issues normally follow

from requirements. Secondly, it is unlikely that a designer would perceive a relationship between an R-node and an I-node that would not correspond to an implementation link, and hence already be included in that form. Of course, the implementation link would represent somewhat different information, but the fact that they join the same nodes means that the implication link would probably have little additional design structuring impact. For these reasons, it is concluded that there is probably little additional R-to-I-node (or vice versa) semantic information that a designer would want, or be able, to include in the design model, that is not captured already in the form of conventional implementation links. Therefore, the alternative of logical implication between R-node and I-node will not be included in the extended design structuring model.

4.4 APPLICATION OF EXTENDED MODEL TO THE 22-NODE DBMS REQUIREMENTS SET.

In order to illustrate an application of the extended architectural design model described in this paper, a set of 22 requirements for a data base management system (DBMS), studied earlier by Andreu (see (Andreu 78)), are analyzed here. The 22 requirements, given in Section 4.4.1 below, are only skeleton requirements for a real DBMS, but they are quite satisfactory for demonstration purposes. In a later report, the extended model will be applied to a larger, more realistic requirements set.

In making the assessments reported here, the following steps were followed.

1. The requirements as used by Andreu were adopted in their entirety. The requirement statements were then transformed into "template form" (see Chapter 3) and the Appendix for more information regarding requirement statement templates).
2. The interdependency assessments made by Andreu in earlier analysis of these requirements were also used here. While this author (and, for present purposes, DBMS designer) does not necessarily agree completely with the appropriateness of the interdependency assessments reported by Andreu, no significant changes were made so as to maintain comparability with the earlier results. The descriptions of these interdependencies, given in Section 4.4.2, were clarified and expanded from their original form.
3. Weights were assigned to each interdependency. The weight interpretation used here was that of strength of interaction, as described in Section 4.3.1. Assigned weights were either W (weak), A (average), or S (strong).

4. Interdependencies were reviewed to determine similarities. This review was conducted as follows: first, an interdependency between, say, requirements n and m was selected. Then, the concept underlying the description of this interdependency was compared, mentally, to the underlying concepts for all other interdependencies in which either requirement n or m was involved. Similarity assessments were rated W, A, or S, similar to interdependency assessments.
5. Requirements were again reviewed pairwise to search for implication relationships, both lateral and hierarchical. Because this experimental requirements' set is quite small, each requirement statement is rather highly abstracted. Hence the likelihood of many logical interactions is small. In fact, only three such relationships were identified.
6. Finally, interdependencies were reviewed pairwise to determine logical relationships. For the same reason as given in (5) above, few such relationships were expected (only one was identified).

The results of these analyses are reported in the following five sections. Section 4.4.1 lists the original DBMS requirements (in template form); 4.4.2 describes the assessed interdependencies and the weights assigned; 4.4.3 gives the interdependency commonality assessments and their assigned weights; 4.4.4 describes implication relationships between requirements; and, 4.4.5 gives the implication relationships between interdependencies.

Figure 4.9 is a schematic representation of the extended model as applied to the 22-node DBMS design problem, using the diagrammatic techniques discussed in Section 4.3. Section 4.4.6 includes comments on the process of mak-

ing the various kinds of assessments demanded by the extended model.

4.4.1 Sample Set of 22 DBMS Requirements.

The requirements given below are slightly modified versions of the 22 requirements analyzed in (Andreu 78).

SAMPLE SET OF 22 DBMS REQUIREMENTS.

1. The database can have multiple logical organizations.
2. There can be user-meaningful logical data-item groups.
3. There can be user-meaningful relationships among data items.
4. There can be algorithmic relationships among data items.
5. There will be logical operations involving data items, groups, and relationships.
6. Data items will be organized physically in a unique way.
7. There will be certain specific queries.
8. Frequencies of queries will be non-uniform.
9. Data items may be referenced according to logical group membership.
10. Data items may be referenced according to item value.
11. Data item retrieval will be as fast as possible.
12. The distribution of data items across queries can be significantly non-uniform.
13. There will be an English-like language for expressing queries.
14. The query language will be unambiguous.
15. Query expressions will be non-procedural.
16. There will be different data item types (e.g., integers, character strings).

17. Same type data items can be combined using certain operations (e.g., addition, for integers; concatenation, for character strings).
18. Certain data items may be represented using alternative data types.
19. There will be a specific value range for each data item.
20. Data items can take values from a subset of their value range.
21. Data redundancy will be minimized.
22. Storage costs will be minimized.

4.4.2 Requirements Interdependencies and Weights.

The assessed interdependencies and assigned weights for the 22 DBMS requirements are given below. The numbers in the "requirement pair" column refer to requirement statements from the previous section. Weight codes are:

W - weak

A - average

S - strong.

INTERDEPENDENCIES BETWEEN DBMS REQUIREMENTS

Requirement Pair	Weight	Interdependency Description
(1,2)	A	Logical views are definable in terms of logical groups
(1,3)	A	Logical views are definable in terms of relationships among data items.
(1,5)	A	Logical operations are carried out in the context of the logical view.
(1,6)	S	Various logical views must be obtainable from a unique physical view.
(1,21)	A	Data redundancy would otherwise be useful in implementing multiple logical views.
(2,3)	W	Relationships may exist both within and between logical groups.
(2,5)	A	Logical groups may participate in logical operations.

- (3,5) S Logical relationships may be involved in logical operations.
- (4,17) A Algorithmic relationships among data items must be consistent with allowable operations.
- (4,18) A Algorithmic relationships must be defined in terms of allowable alternative data types.
- (4,21) A Depending on how implemented, algorithmic relationships can help to avoid redundancy.
- (4,22) S By virtualizing certain data, storage requirements may be reduced through use of algorithmic relationships.
- (5,7) S Queries must be computable using defined operations.
- (5,15) A Mapping(s) must exist between non-procedural queries and the set of logical operations.
- (6,9) W Logical group membership should unambiguously correspond to membership in some part of the unique physical organization.
- (6,21) W A unique physical organization favors non-redundancy.
- (7,13) W All queries should be expressible in the English-like query language.
- (7,14) W All queries should be unambiguous.
- (7,15) W All queries should be expressible in non-procedural fashion.
- (8,11) S Data physical organization should reflect query type frequency to minimize lookup time.
- (8,12) A Frequencies of queries and frequencies of data items within queries determine frequencies of data items' references.

- (9,11) S Data item location via logical group membership can affect query response time.
- (9,12) W Database searching mechanisms should take into account distribution of data items in logical groups.
- (9,21) A Representing every logical group physically is an alternative that goes against avoiding redundancy.
- (10,11) V Alternative mechanisms for locating data items by value have efficiency implications.
- (10,12) A Alternative mechanisms for locating data items by value should take into account their frequency of reference in queries.
- (10,19) A Bounds checking can be used to resolve references by data type.
- (10,20) W A data reference value may fall within the relevant range for that data item, yet not be in the data base.
- (11,19) A Bounds checking can enhance response time for certain queries.
- (11,20) W Knowledge of relevant target values may help in choice of most efficient search strategy.
- (11,22) S More efficient lookup strategies usually require more memory.
- (13,14) W The more English-like a query language, the greater the scope for ambiguous queries.
- (14,15) W The less procedural the query language, the greater the opportunity for ambiguity.
- (16,17) A Operations must be consistent with the type of data item upon which they act.

- (16,18) A as for (16,17)
- (16,22) W Certain data types may be stored more compactly than others.
- (17,18) S Operations must always be consistent with the data types used.
- (18,22) A Certain data types may be stored more compactly than others.
- (21,22) A Decreased redundancy means decreased storage costs.

4.4.3 Interdependency Similarity Assessment.

Assessed similarity relationships and weights between pairs of interdependencies are given below. The numbering scheme for interdependency pairs refers to the original requirements' numbers. For example, (1,2) corresponds to the interdependency between requirements 1 and 2. The weight codes used are the same as used for interdependency weights (see Section 4.4.2).

INTERDEPENDENCY SIMILARITY ASSESSMENTS

Interdependency Pair	Weight	Description of Nature of Similarity
(1,2), (1,3)	A	Common issue regarding definition of logical groups.
(2,5), (3,5)	A	Participation in logical operations.
(4,17), (4,18)	S	Common virtualization issue.
(4,21), (4,22)	A	Common virtualization issue.
(10,11), (10,12)	A	Similar efficiency issues.
(10,11), (10,19)	A	Both related to value representation.
(10,19), (10,20)	A	Both related to data reference by value.
(13,14), (14,15)	S	Both concern query language design.
(16,17), (16,18)	S	Same issue.
(16,22), (18,22)	S	Both related to data type selection issue.

4.4.4 Implication Relationships Between Requirements.

Three implication relationships between requirements were determined, shown below. The first two are one-way lateral implications, the third is a two-way lateral implication. No hierarchical implication relationships were detected in the 22-node DEMS set.

IMPLICATION RELATIONSHIPS BETWEEN REQUIREMENTS

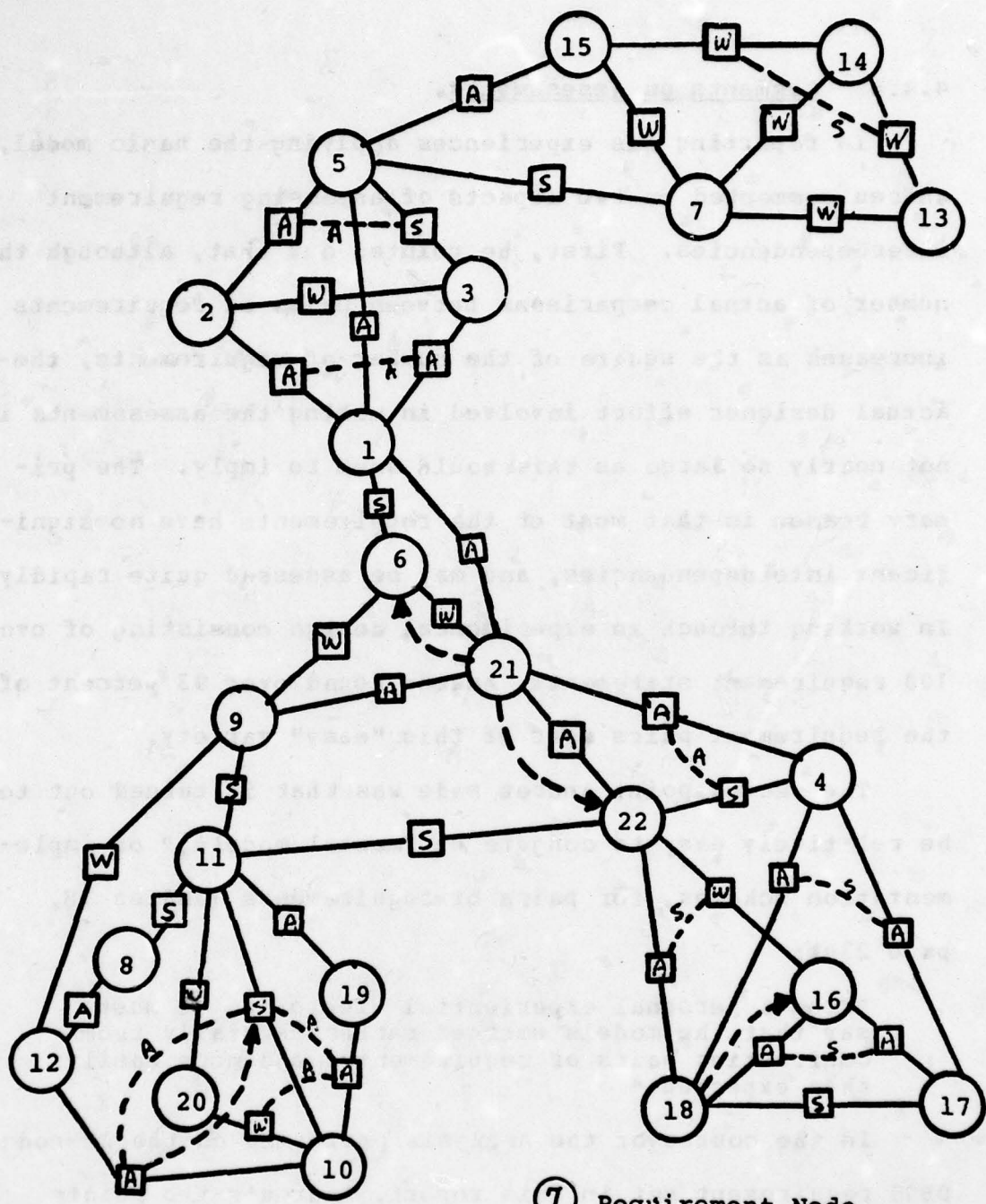
Requirement Pair	Nature of Relationship	Comments
6,21	21 --> 6	To minimize data redundancy, the system ought to support a unique physical organization of data.
16,18	18 --> 16	In order to be able to represent certain items using various data types, the system must support an appropriate variety of data types.
21,22	21 --> 22	Reducing data redundancy tends to reduce storage costs.

4.4.5 Implication Relationships Between Implementation Issues.

A single one-way implication relationship between interdependencies was assessed, as given below.

IMPLICATION RELATIONSHIPS BETWEEN INTERDEPENDENCIES

Interdependency Pair	Nature of Relationship	Comments
(10,11), (10,12)	(10,12) --> (10,11)	An implementation of a mechanism for locating data items by value that takes into account the items' distribution across queries will generally also be more efficient in terms of retrieval response.



⑦ Requirement Node

—[A]— Implementation link,
node, and weight

□--[A]--□ Similarity link and
weight

- - - → Implication link

Figure 4.9

Graph diagram of DBMS design problem using extended model

4.4.6 Comments on Assessments.

In reporting his experiences applying the basic model, Andreu commented on two aspects of assessing requirement interdependencies. First, he pointed out that, although the number of actual comparisons between pairs of requirements increases as the square of the number of requirements, the actual designer effort involved in making the assessments is not nearly so large as this would seem to imply. The primary reason is that most of the requirements have no significant interdependencies, and may be assessed quite rapidly. In working through an experimental design consisting of over 100 requirement statements, Andreu found over 93 percent of the requirement pairs were of this "easy" variety.

The second point Andreu made was that it turned out to be relatively easy to conjure up "mental models," or implementation schemes, for pairs of requirements (Andreu 78, page 234):

"From a personal experiential viewpoint, we must say that the models emerged rather naturally from confronting pairs of requirements, and more easily than expected."

In the course of the analysis performed on the 22-node DBMS requirement set in this report, Andreu's two points were found to hold. Also, experience gained in the determination of the additional kinds of information - interdependency weights, interdependency similarities and associated

weights, and implication relationships - is summarized as follows:

4.4.6.1 Weight Assessments.

It was earlier suggested by Andreu that it might be possible to make weight assessments by counting the number of different implementation schemes that underly a given interdependency, then assigning a weight value based on the total number of such schemes. This approach was not found to be very useful, for two reasons.

First, most of the related requirement pairs were actually related via a single interdependency. However, in a number of cases, the strengths of these single interdependencies were judged significantly different, hence deserved different weight values. Andreu's proposed approach would not properly handle this situation.

The other main reason is that many of the conceptual models of interdependence were sufficiently general in nature that it wouldn't be meaningful to try to "count" them as individual implementation schemes.

Rather than attempt to use a mechanical approach to determining the weights to be assigned to each interdependency, the assessment was made judgmentally, i.e., by men-

tally examining the same conceptual models used in determining the interdependencies themselves in the first place. It must be granted that such a judgmental approach to eliciting the strengths of interdependencies would probably lead to a fairly high variability among different designers in practice. However, this variability could be reduced somewhat, possibly using a Delphi-like technique which would have the designer re-think his assessments in light of assessments made by other designers.(15) Also, some variability among different designers is to be expected, inasmuch as the different designers perceive the many design issues in different ways.

The above comments also apply to assessment of the strengths of similarity relationships between implementation considerations.

4.4.6.2 Implication Relationship Assessments.

Few implication relationships were detected in the set of 22 DBMS requirements. The main reason for this is that the requirement statements were few in number and broad in scope. It is expected that a more realistic (i.e., larger)

(15) There are many similarities between the Delphi technique used in forecasting and "opinion averaging" analysis, and some of the modern programming management methods such as "structured walkthroughs."

requirements set would exhibit a higher proportion of requirement and implementation issue implication relationships.

Those relationships that were assessed in the 22-node requirement set were actually identified rather easily. The scanning of requirements statements or interdependency description statements to locate implication relationships could be done rapidly, and related statements generally stood out quite plainly.

On balance, the additional assessment time and effort needed to determine the extra information required by the extended model was somewhat less than that needed to determine the original basic interdependencies. The demands that would be placed on a designer to supply this data are non-trivial, but not out of the realm of reason, assuming the designer believes that the final quality of the design structure will be significantly improved as a result.

4.5 SUMMARY.

The argument in this chapter has been that there are important additional kinds of information designers generally possess that cannot be represented in the basic architectural design framework developed originally by Andreu. Certain classes of such untapped information, believed to be most relevant and accessible via designer judgment and knowledge, have been identified. Possible schemes for representing much of this additional information in the context of an extended graph model have been discussed.

To summarize the various kinds of information proposed for the extended architectural design framework, consider Figure 4.10. Shown here is the schematic representation of certain design information, using the basic design model, involving five functional requirements (1,2,3,4, and 5). A designer, having studied the 10 requirement pairs, has identified five implementation interrelationships (1-2, 1-3, 2-3, 1-4, and 1-5).

Adding extension 1 to the representation - weights on the implementation links - the designer's assessments might be as shown in Figure 4.11.

Extension 2 is then added: the implementation issues are explicitly represented as I-nodes, and similarities among them are determined and added to the diagram. Assuming a single such similarity relationship, suppose implemen-

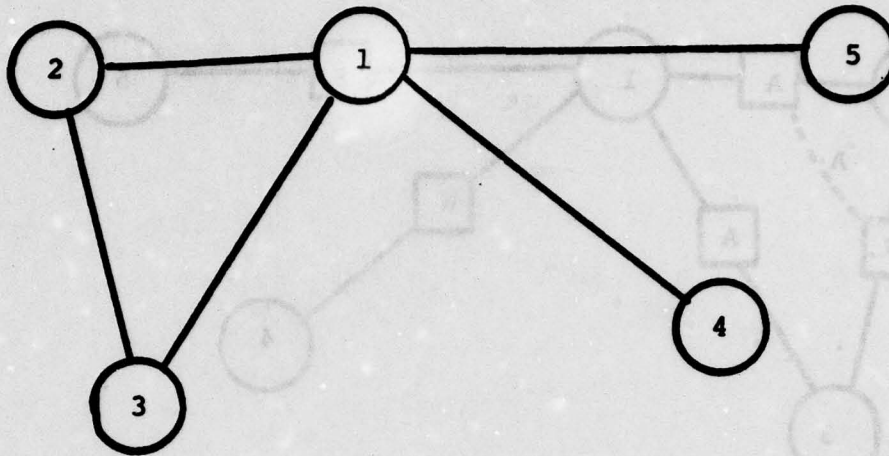


Figure 4.10

Basic requirements graph model

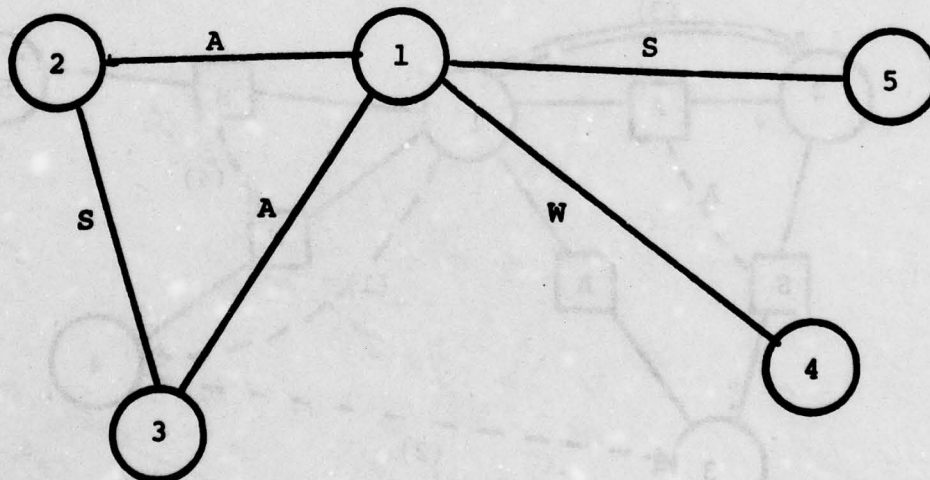


Figure 4.11

Interdependency weights added to model of Figure 4.10

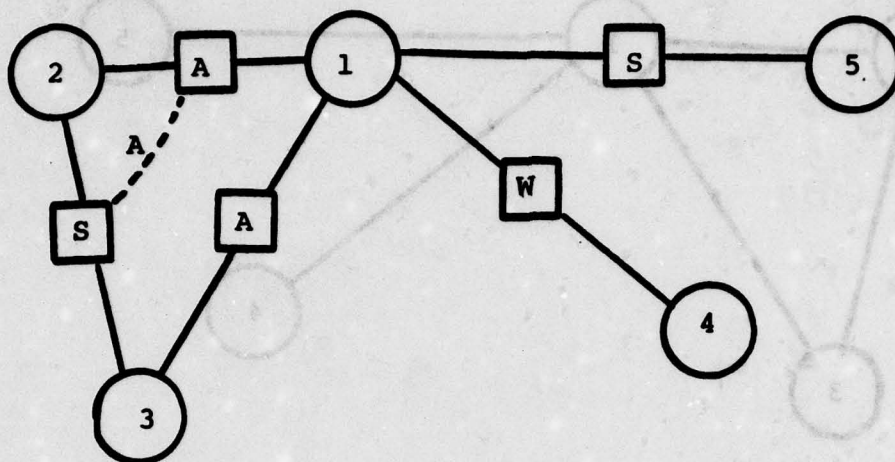


Figure 4.12

Interdependency similarity information added to model of Figure 4.11

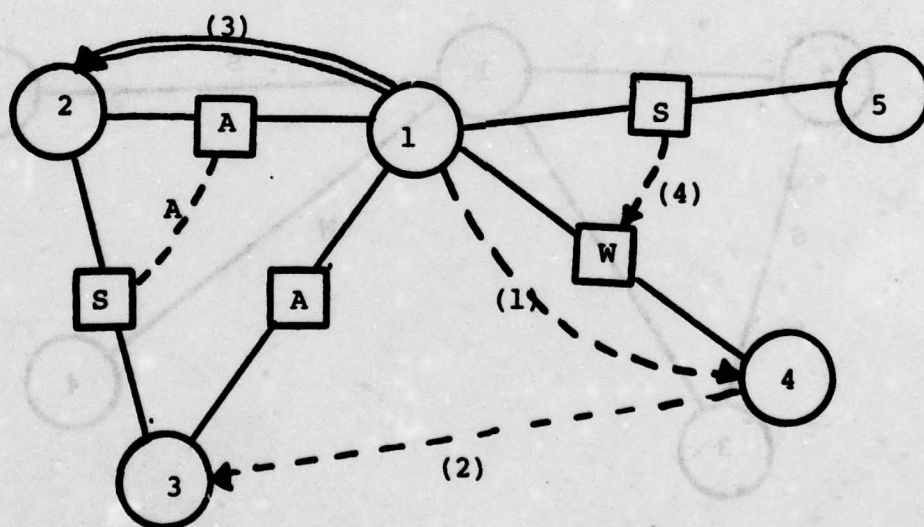


Figure 4.13

Implication information added to model of Figure 4.12

tation nodes (1-2) and (1-3) are determined to represent similar issues (within estimated 50 percent average overlap). This information is portrayed in Figure 4.12.

Finally, additional semantic information, in the form of logical implications at both functional level and implementation level are assessed and added to the schematic. In Figure 4.13, (1) represents a lateral implication from requirement 1 to requirement 4; (2) represents a bi-directional lateral implication between requirements 3 and 4; (3) is a hierarchical functional implication from 1 to 2; finally, (4) is a logical implication relationship between implementation issues (1,4) and (1,5).

It was not the intention of the discussion in this chapter to imply that all alternatives under consideration have to be brought to bear in modelling design structuring information - only that they might be useful. Specifically, ways in which such information could be incorporated into design structuring algorithms will be the subject of the next chapter.

Chapter V

GRAPH DECOMPOSITION ANALYSIS TECHNIQUES FOR USE WITH THE EXTENDED SDM MODEL.

5.1 INTRODUCTION.

In his initial research on software architecture, Andreu employed a simple binary, undirected graph model to represent a system's requirements and their implementation interdependencies. While this "bare bones" model proved satisfactory for the early exploratory studies, it was also clear that improvements, primarily in the form of extensions, could be made so as to allow a designer to represent additional design-relevant information.

Certain of these potential extensions were identified and discussed in the previous chapter. There it was argued that the most significant such extension was the inclusion of a weight factor to correspond to each assessed interdependency. With this extension, the requirements graph becomes a weighted graph, with a weight on each arc representing the strength of the corresponding interdependency. Other possible extensions were also discussed there, including relationships between interdependencies, as well as certain kinds of directed relationships.

An important part of the Systematic Design Methodology is the set of analysis techniques that are used to perform various kinds of decomposition analysis on a given requirements graph. Included here are procedures for calculating the goodness index for a given graph partition (based upon the "strength/coupling" criterion espoused by a number of software design theorists; see, for instance, (Stevens, et. al. 75)); procedures for calculating similarity and/or distance measures between pairs of for subsequent use in clustering algorithms; clustering techniques themselves, for performing hierarchical cluster analysis to generate graph decompositions; and other types of decomposition analysis routines, including a new top-down hierarchical partitioning algorithm developed especially for treating the SDM graph decomposition problem (discussed in detail in the next chapter).

Having extended the SDM representational framework, it becomes necessary to analysis techniques so as to incorporate the information included in the new representation. The major purpose of this chapter is to present, justify and discuss certain new analytical mechanisms that were developed to treat the requirements decomposition problem in the context of the extended SDM model. In addition, this chapter will present other new analytical techniques and discuss their pros and cons. Some potentially valuable decomposi-

tion approaches, which have not yet been fully exploited in the SDM context, will be outlined. The results of a comparative analysis among currently viable decomposition methods will be presented. Two different approaches to incorporating interdependency similarity information into the graph decomposition will also be briefly discussed. A medium-scale example analysis illustrating the effectiveness of the extended analysis techniques is presented, and contrasted to the somewhat different results obtained for the same example graph in earlier work.

Finally, appendices D and E include documentation and an example execution trace of the computer package that has been developed to implement the new analysis methods.

5.2 IDM ANALYSIS AND INTERDEPENDENCY WEIGHTS.

As stated above, the single most useful and important extension made to the original requirements graph model involves incorporation of weight factors corresponding to the requirement interdependencies. In our work to date, weights are chosen from the range [0, 1.0], with lower values corresponding to "weaker" interdependencies. A useful practical device in this regard is to select the weights from three possible candidates: W (weak), A (average), or S (strong). For computational purposes, these are mapped into numerical values, for example,

S - 0.8

A - 0.5

W - 0.2 .

In this section we examine extensions that have been made to various analysis mechanisms to incorporate such link weights.

5.2.1 Extension to Decomposition Goodness Index.

The concept of decomposition goodness has been captured by quantifying a commonly accepted notion of software design quality: Alexander (Alexander 64), Stevens (Stevens 75), MYERS (MYERS 78) and others have convincingly argued that a good software design is one that consists of modules that possess high strength, or internal binding, and which simul-

taneously are weakly interconnected. In the SDM, this "strength/coupling" criterion is quantified in the following way. Suppose the graph representation of the target design problem has been decomposed into a set of non-overlapping subgraphs:

$$\{G_1, G_2, \dots, G_k\}.$$

Then, if S_i = the strength of subgraph G_i , and C_{ij} = the coupling between subgraphs G_i and G_j , we define

$$M = \sum_{i=1}^k S_i - \sum_{i=1}^{k-1} \sum_{j=i+1}^k C_{ij}$$

and use M as a figure of merit for the decomposition.

5.2.1.1 Strength and Coupling - Unweighted Graphs.

The quantities S_i and C_{ij} are themselves defined in terms of the structure of the corresponding subgraphs. Various arguments regarding how S_i and C_{ij} ought to be defined in the case of the original graph model (with unweighted links) are discussed by Andreu (Andreu 78), and will not be repeated here. The following definitions for these quantities were given:

(1) define S_i (strength of subgraph i) as:

$$S_i = [L_i - (n_i - 1)] / [n_i (n_i - 1) / 2]$$

AD-A074 911

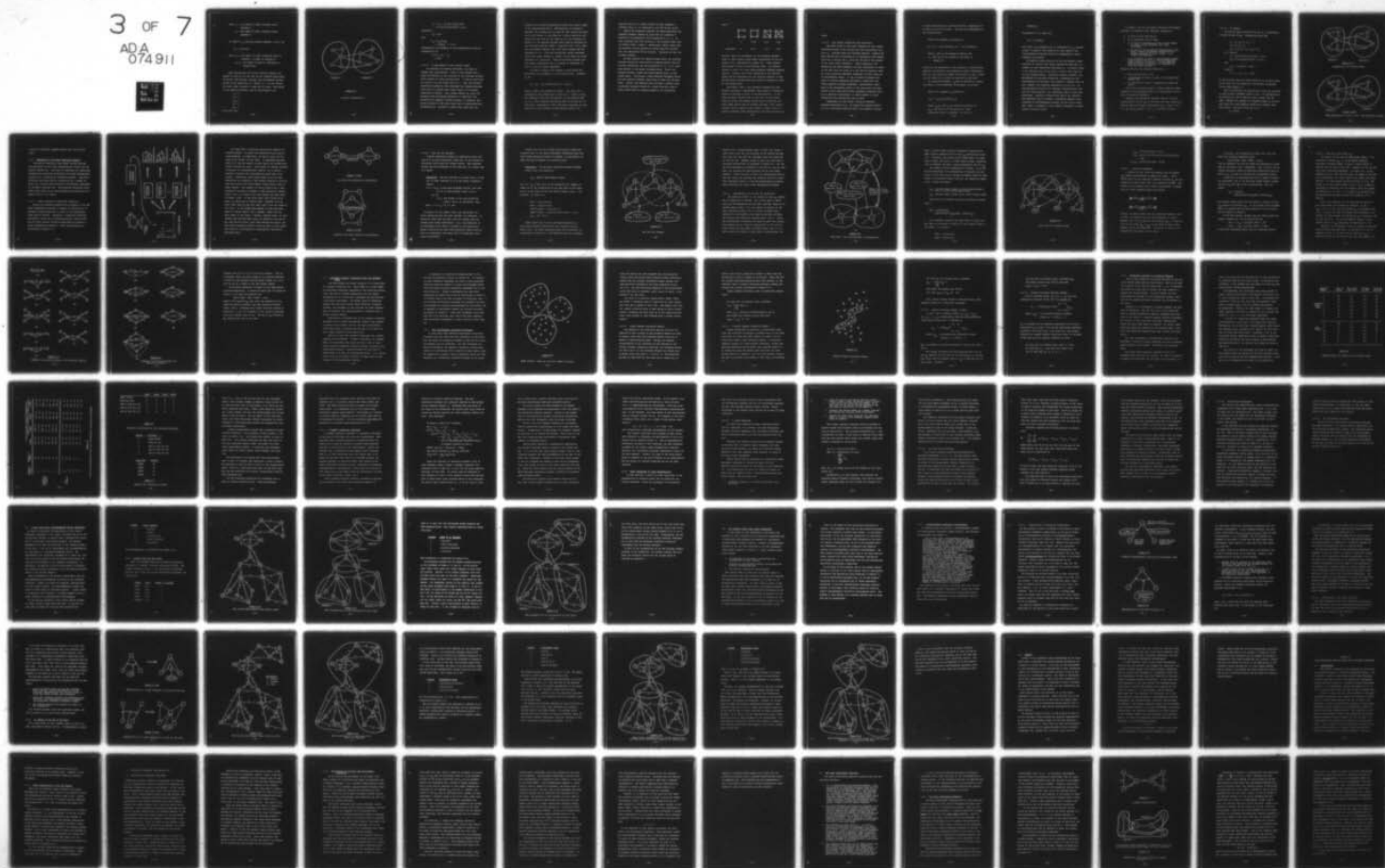
ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC METHODOLOGY FOR DESIGNING THE ARCHITECTURE OF COMP--ETC(U)
SEP 79 S L HUFF, S E MADNICK
CISR-P010-7906-12

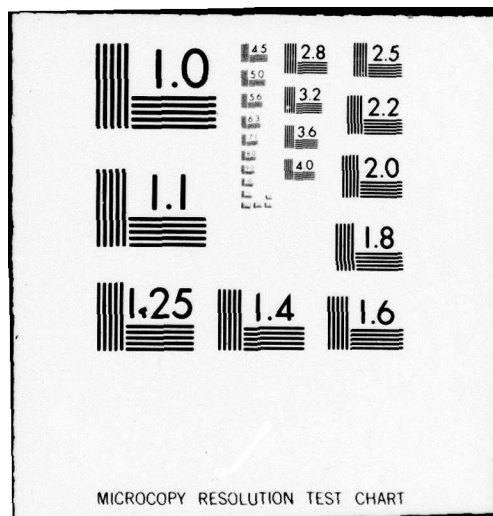
N00039-78-G-0160

NL

UNCLASSIFIED

3 OF 7
ADA
074911





where L_i = the number of links contained within
subgraph i ,

n_i = the number of nodes contained within
subgraph i .

(2) define C_{ij} (coupling between subgraph i and j) as:

$$C_{ij} = L_{ij} / (n_i n_j)$$

where L_{ij} = the number of links connecting nodes in
subgraph i to nodes in subgraph j ,

n_i, n_j = the number of nodes in subgraphs i, j
respectively.

These definitions have strong intuitive appeal, and seemed to work well in the case of the original graph model. To clarify them further, consider the two-subgraph decomposition of the graph shown in Figure 5.1. In that figure, the total graph includes 11 nodes and 16 links. The values of the various parameters used in calculating M are:

$$L_1 = 6$$

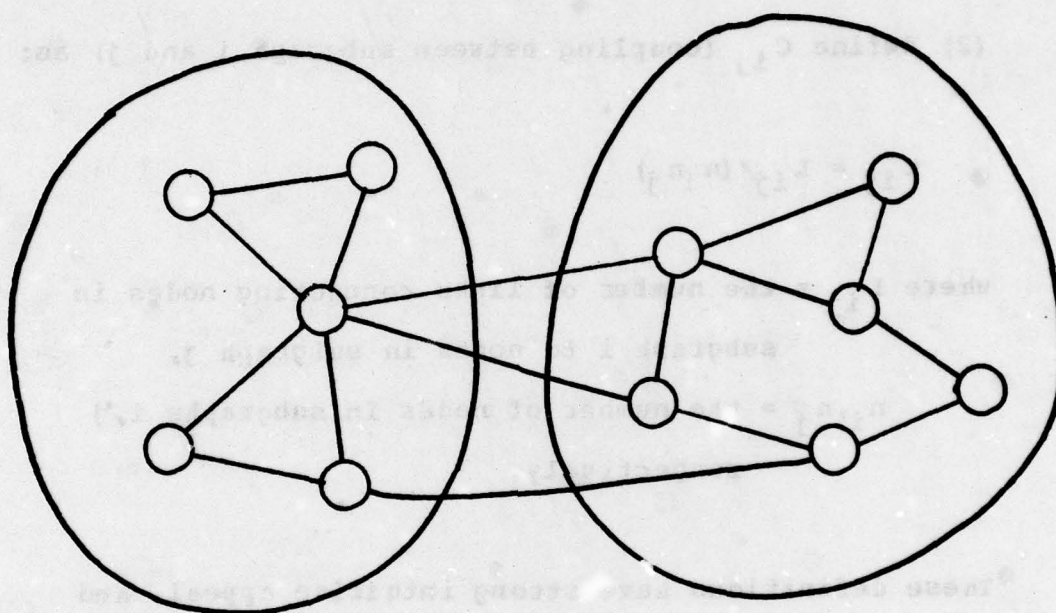
$$L_2 = 7$$

$$L_{12} = 3$$

$$n_1 = 5$$

$$n_2 = 6$$

So we find that



Subgraph 1

Subgraph 2

Figure 5.1

A simple decomposition.

$$S_1 = (L_1 - (n_1 - 1)) / (n_1 (n_1 - 1) / 2) \\ = (6 - (5 - 1)) / (5(5 - 1) / 2) = 0.20 .$$

Similarly,

$$S_2 = 0.13 .$$

And,

$$C_{12} = L_{12} / (n_1 n_2) \\ = 3 / (5(6)) = 0.10 .$$

Consequently, the goodness of this decomposition would be calculated to be

$$M = S_1 + S_2 - C_{12} \\ = 0.230$$

5.2.1.2 An Improvement to the Strength Index.

In extending the measure definition, two kinds of changes were incorporated. First, it was decided that a small modification to the structure of the strength function would improve its value qualitatively. It may be noted that the value of C_{ij} can range from 0 to 1. Now, the software engineering literature that addresses the strength/coupling issue does not suggest that either factor is of primary importance in design. Also, Andreu (Andreu 78, page 104) presented some motivating arguments to illustrate that weighting one component (either strength or coupling) unequally relative to the other could lead to counterintuitive decompositions. Therefore it makes most sense that our

strength and coupling formulations should carry equal weight in the determination of M . Unfortunately, as presently defined, the strength term S_i does not fall within the range $[0,1]$, but rather, in the range $[0, 1-2/n_i]$, while the coupling term does fall in the range $[0,1]$. For instance, in Figure 5.1, the maximum strength that could be exhibited by the five-node subgraph number 1 would be $1-2/5 = 0.6$, whereas the maximum coupling that could occur between the two subgraphs is 1.0. From this definition, larger subgraphs would have higher maximum S_i values, as the term $2/n_i$ would decrease as n_i increases. These observations suggest that the present formulation for S_i includes an (unwanted) an bias in favor of larger subgraphs.

In order to remedy this problem, we may adjust the definition of S_i slightly, in the following way. Redefine S_i as:

$$S_i = [L_i - (n_i - 1)] / [n_i(n_i - 1)/2 - (n_i - 1)]$$

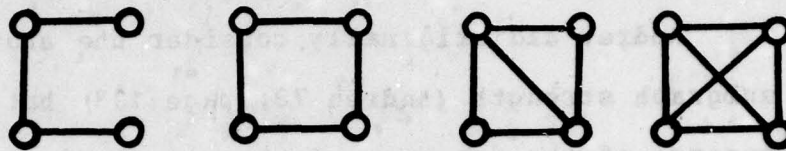
where L_i and n_i are defined as before. The range of S_i according to this definition is then $[0,1]$. There is only one difficulty with this definition: S_i is undefined when $n_i = 2$. Thus a special calculation must be carried out in this case. Fortunately, since very small subgraphs are generally of little interest in SDM analysis anyway, the

approach taken is to simply assign two-node subgraphs a strength value of 1.0 (modified by the link weight factor,

Andreu did originally consider the above definition for subgraph strength (Andreu 78, page 103) but rejected it because of the presence of the singularity at $n_i = 2$. He also commented that both versions of the strength index tend to produce similar results. Essentially, Andreu argued that the modified version appeared no better than the original one, and complicated things slightly. Therefore he kept the original version.

We take somewhat the opposite stance here: the modified strength definition given above is arguably better than the original definition, and the singularity at $n_i = 2$ is trivially avoided in the manner discussed above. First, the modified strength avoids the large-subgraph bias, as discussed above. Intuitively, fully-connected subgraphs should all have maximum strength, which they do under the modified index. Second, the modified index provides a greater range of possible strength values for a given node set, hence a higher sensitivity to subgraph geometry, as illustrated

below:



S	0	0.167	0.33	0.50
---	---	-------	------	------

modified S	0	0.33	0.67	1.00
------------	---	------	------	------

Thirdly, there are precedents for the modified strength index in other similar graph model applications in the literature (e.g., Estabrooke 66; Hubert 74). These authors have argued in favor of the modified index, in contexts similar to ours, as a good general-purpose subgraph strength measure. Finally, the "nice" properties of the modified index, and the parallels with the coupling measure, as summarized below, provide additional indirect evidence in favor of the modification.

Admittedly, none of the foregoing arguments is completely conclusive. Nonetheless, they present a cumulative weight of evidence in favor of the modification. In the final analysis, choices such as this one, in the present research effort, are perhaps guided more by intuition and "what makes sense" than by provable theorems. (This characteristic is not unique to SDM, either.) While trying to locate an example graph decomposition that would "prove" the

index.

5.2.1.3 Link Weight Information and Similarity.

The other issue at this point concerns how link weight information ought to be factored into the calculation of S_i and C_{ij} . Consider first the strength function. Perhaps the most obvious way of extending S_i to incorporate link weights would be to replace the L_i term with the sum of the weights on the links within subgraph i . While appealing, this definition has some drawbacks, the most significant of which is the fact that the value of S_i may then be negative, even for fully connected subgraphs (subgraphs in which there are no disconnected nodes). It may of course be argued that there is nothing especially bad about a definition that admits negative strength subgraphs. Nevertheless, since most of our development effort in the SDH project has been guided by what seems intuitively reasonable (lacking criteria of absolute correctness), it seems prudent to avoid counterintuitive definitions such as this.

Consequently we will adopt a slightly different extended definition of S_i . We accept the original definition as a reasonable starting point, and attempt to extend

it while maintaining its positive features, especially its residing in the $[0,1]$ range. This may be accomplished in the following way:

Define the extended S'_i function as

$$S'_i = [L_i - (n_i - 1)] / [n_i(n_i - 1) / 2 - (n_i - 1)] * (W_i / L_i)$$

where L_i and n_i are defined as before, and

W_i = the sum of the weights on the links in subgraph i .

S'_i has all the properties of the original S_i , plus the new property of reflecting interdependency weight information. That is, the higher the summed link weights W_i (for a given L_i) the higher S'_i , as would be required by intuition.

In a parallel fashion, C'_{ij} may be modified to capture the effect of inter-subgraph link weights, as follows.

Define the extended C'_{ij} function as

$$C'_{ij} = [L_{ij} / (n_i n_j)] * (W_{ij} / L_{ij})$$

where L_{ij} , n_i , and n_j are defined as before, and

W_{ij} = the sum of the weights on the links connecting nodes in subgraph i to nodes in

subgraph j.

Consequently it is clear that

$$C'_{ij} = w_{ij} / (n_i n_j)$$

Once again, C'_{ij} possesses all the properties of C_{ij} (specifically, it falls in the range $[0,1]$, plus captures the interdependency strength effects as represented by inter-subgraph link weights.

The question again arises as to why the foregoing modifications to capture link weight information were used, as opposed to some alternative modifications. As before, there is no "theorem" that can be used to "prove" that these are the best modifications. Cumulative indirect evidence (e.g., maintenance of desirable properties, widespread use and understanding of arithmetic mean concept), together with our best judgment and favorable experience to date (i.e., no counterintuitive results for "obvious" decompositions) support the above choices. One additional point is also worth mentioning. Empirical evidence with SDM so far indicates that designers tend to produce a reasonably symmetric distribution of interdependency weights, so that using arithmetic mean, as opposed to, say, median, introduces no appreciable "long tail" bias.

In summary, the new (extended) definition of subgraph strength, S'_i , exhibits three important properties:

1. it falls in the range $[0, 1]$;
2. it is normalized, in two ways:
 - a) in terms of subgraph size (for a given number of links, larger subgraphs have lower strengths),
 - b) in terms of "tree-relative connectedness" (subgraph strength is measured relative to the minimum necessary to form a graph - i.e., "tree" connectedness);
3. it is invariant in terms of "proportional connectedness": regardless of n , a tree-connected subgraph always has strength 0, a fully connected subgraph always has strength 1.0 (assuming all links have unity weight).

Similarly for the new coupling definition C :

1. falls in the range $[0, 1]$,
2. is normalized in terms of size of the coupled subgraphs, and
3. is invariant in terms of "proportional connectedness."

The strong intuitive appeal of these properties lends credence to the appropriateness of the definitions of subgraph strength and coupling.

Finally, one additional important feature of these definitions of S'_i and C'_{ij} is the fact that they include as a special case the original definitions (i.e., weight set to 1.0, S'_i and C'_{ij} become S_i and C_{ij} as defined earlier).

5.2.1.4 An Example.

To see how these functions work out in a calculation, consider Figure 5.2(a). Computations show that

$$L_1 = 6, L_2 = 7, L_{12} = 3$$

$$n_1 = 5, n_2 = 6,$$

$$W_1 = 3.2, W_2 = 3.8, W_{12} = 1.2$$

As a result,

$$S_1 = (6-4)/(5(4)/2-4) * 3.2/6 = 0.18 ,$$

$$S_2 = (7-5)/(6(5)/2) * 3.8/7 = 0.12 ,$$

and

$$C_{12} = 1.2/(5(6)) = 0.04 .$$

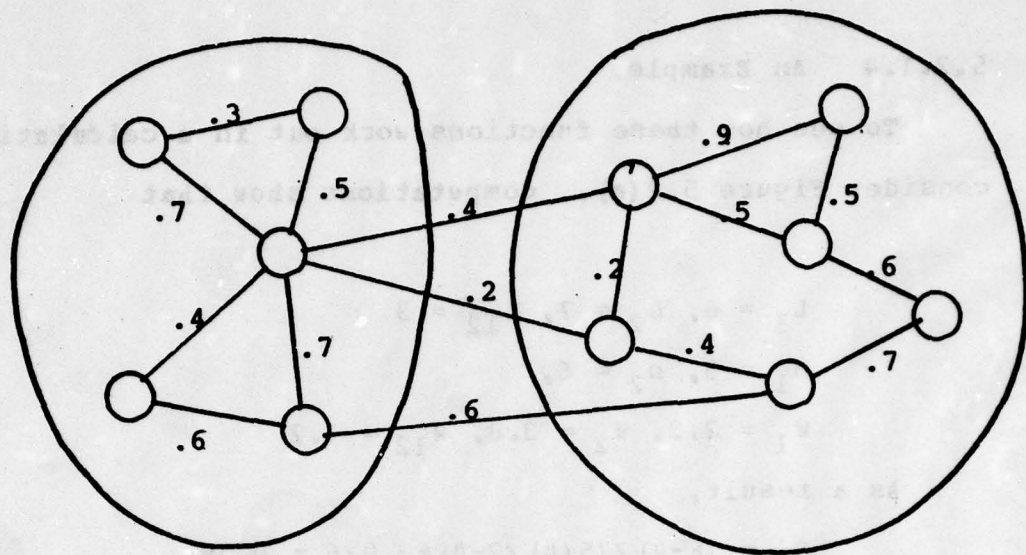
Finally,

$$M = S_1 + S_2 - C_{12} = 0.26 .$$

It may be noted that this value turns out to be quite close to the value of 0.23 obtained in the earlier (unweighted links) case, Figure 5.1.

Now, to illustrate the sensitivity of the new functions to link weights, consider Figure 5.2(b). The decomposition is identical to that of Figure 5.2(a), except that the inter-subgraph link weights are slightly higher on the average, while the intra-subgraph weights are slightly lower than before. The new M turns out to be

$$M = 0.318,$$

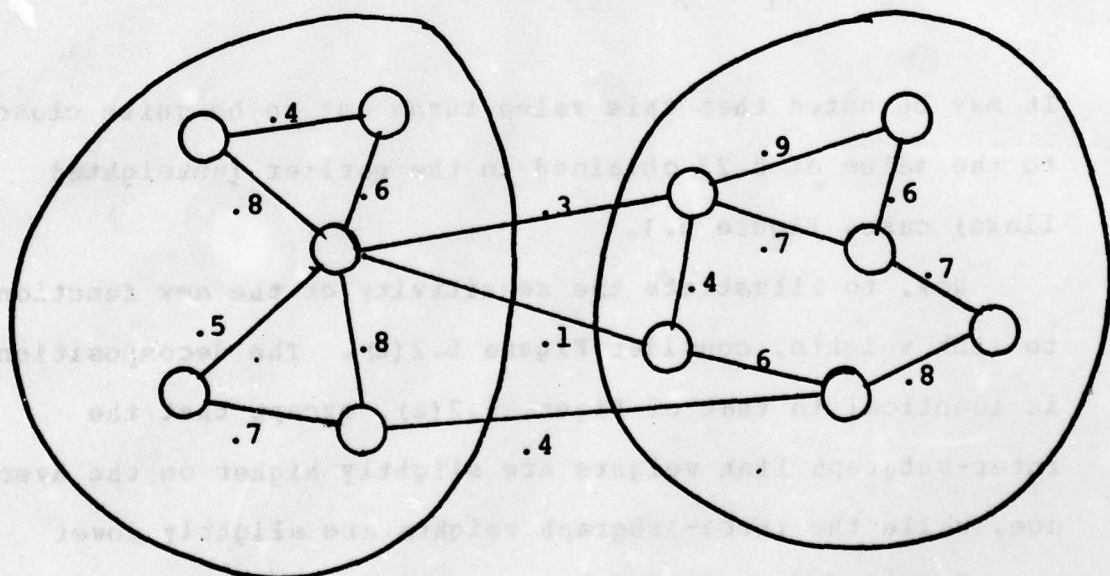


Subgraph 1

Subgraph 2

Figure 5.2(a)

Decomposition of weighted graph.



Subgraph 1

Subgraph 2

Figure 5.2(b)

Same decomposition as for 5.2(a), with different weights

a value, as expected, somewhat higher than the previous value.

5.2.2 Extensions to the Basic Similarity Measure.

The central analytical task within the SDM involves identification of good graph decompositions (those with the highest possible M). One class of techniques for generating decompositions involves transforming the graph decomposition problem into a hierarchical clustering problem. Clustering techniques have been studied intensively for a number of years, and a rather large collection of different algorithms is available (Hartigan 76). Some specific clustering algorithms found useful in the SDM decomposition problem are discussed later in this report.

5.2.2.1 Basic Concepts of Inter-node Similarity.

Before any clustering methods may be applied to the SDM graph decomposition problem, a measure of "similarity," or closeness between each pair of nodes in the requirements graph must be defined. Basically, a similarity algorithm transforms a graph into a similarity matrix, which may be used to drive various clustering algorithms to produce a graph decomposition hierarchy. These relationships are illustrated in Figure 5.3.

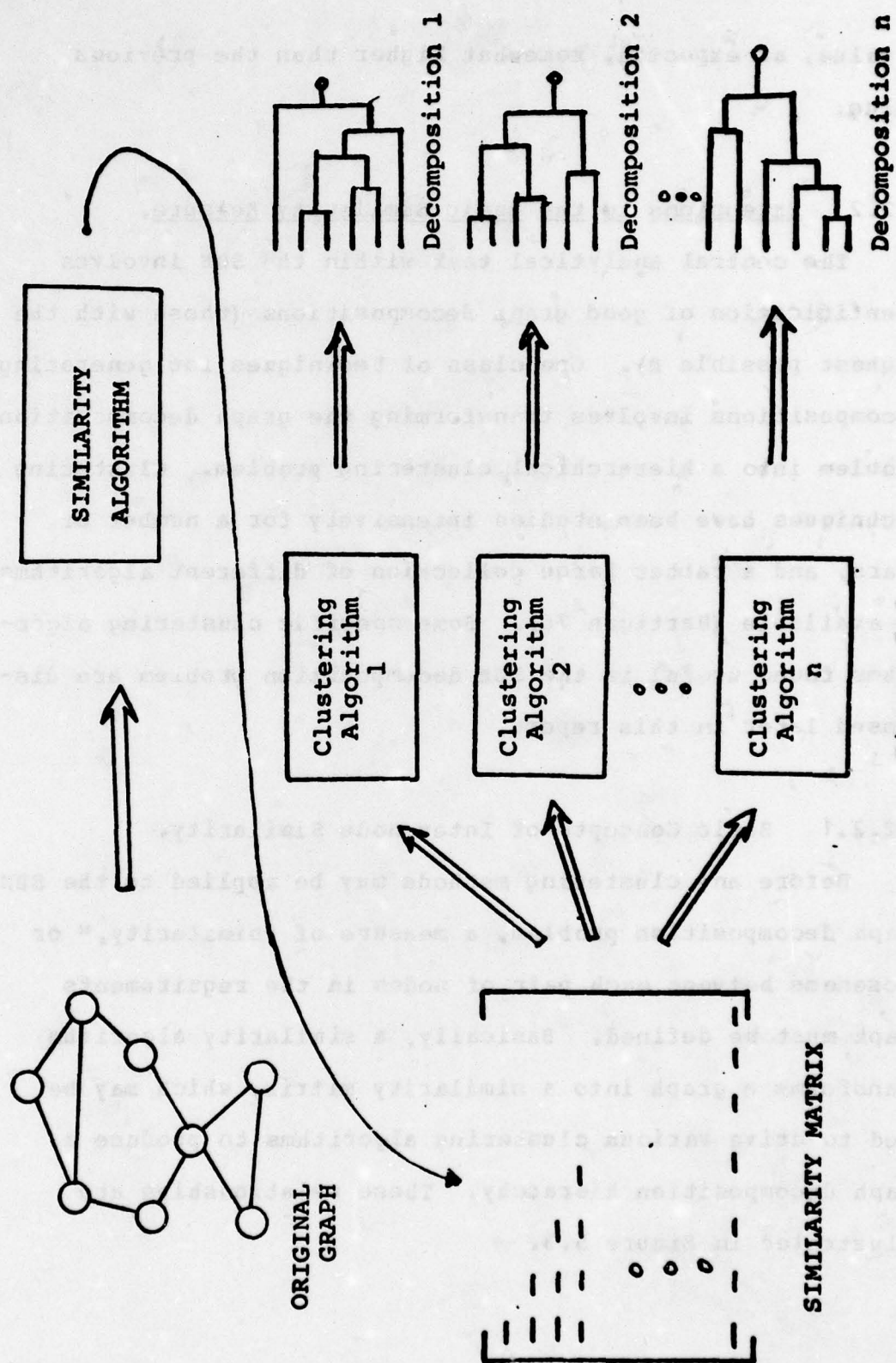


Figure 5.3

Graph decomposition via cluster analysis - the process.

How ought such a similarity algorithm be defined for weighted graphs? To answer this question we must have an interpretation, in graph terms, of what it means for two nodes to be "close" to each other. A reasonable approach might be to use the concept of path length (or minimum path length) between two nodes. Path length, of course, produces a distance (or "dissimilarity") measure, so to obtain a similarity measure some transformation would have to be applied. Regardless, path length proves to be not very appropriate as a measure in this context because it fails to take into account the "environment" within which a pair of nodes resides. For example, we would clearly want a good similarity algorithm to produce a lower similarity between nodes x and y in Figure 5.4(a) than between the same nodes in Figure 5.4(b). In the first case, other things being equal, we would want to cluster node x together with the other three nodes on the left side, node y with the nodes on the right. In the second case however it would make most sense to cluster nodes x and y together, along with the other nodes in the figure. However, assuming equal x-y link weights (and assuming path length is defined as the sum of the link weights along a given path through the graph), the path length similarity measure would be equal in both cases, thus would not distinguish the environmental differences mentioned above.

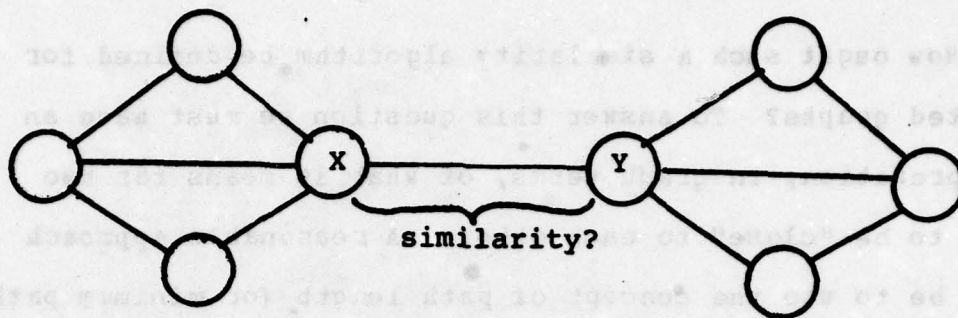


Figure 5.4(a)

An inter-node similarity illustration.

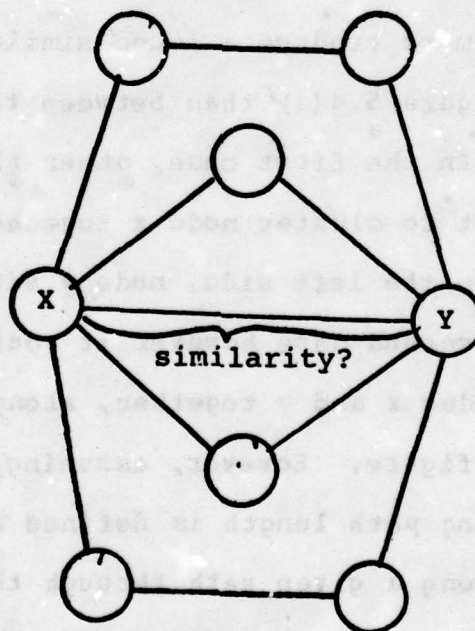


Figure 5.4 (b)

A second inter-node similarity illustration.

5.2.2.2 The Core Set Approach.

A better approach is based on a definition first suggested by Gottlieb (Gottlieb & Kumar 68), in the context of clustering index terms for library design. This approach begins with the definition of the "core set" of a given node in a graph:

Definition. The core set $\phi(x)$ of a given node x is the set of nodes connected to x in the graph, including x itself.

If $A = (a_{ij})$ is the graph adjacency matrix, such that

$$a_{ij} = \begin{cases} 0 & \text{if no links connect nodes } i \text{ and } j, \\ w_{ij}, & \text{the weight on the link connecting} \\ & \text{nodes } i \text{ and } j, \text{ if one exists, then} \end{cases}$$

$$\phi(x) = \{ y: a_{ij} > 0 \} \cup \{x\}.$$

In Figure 5.5, two nodes x and y are identified, and their associated core sets $\phi(x)$ and $\phi(y)$ are indicated. It should be noted that the basic core set concept does not include link weight information - i.e., link existence or non-existence is all that is involved in the definition. Hence a definition of inter-node similarity based solely on core set information will again fail to incorporate link weight information.

However the core set concept does provide a means for including part of the graph environment information that the path length definition failed to achieve. In particular, we adopt Gottlieb's measure as a starting point:

Definition. Let the basic similarity measure between nodes x and y be defined as

$$P_{xy} = |\mathcal{C}(x) \cap \mathcal{C}(y)| / |\mathcal{C}(x) \cup \mathcal{C}(y)|$$

That is, P_{xy} is the ratio of the cardinality of (number of nodes in) of the intersection of the core sets to the cardinality of the union of the core sets of nodes x and y . For instance, in Figure 5.5,

$$\mathcal{C}(x) = \{x, 2, 3, 4, 5, y\}$$

$$\mathcal{C}(y) = \{y, x, 5, 6, 7, 8\}$$

$$|\mathcal{C}(x) \cap \mathcal{C}(y)| = |\{x, y, 5\}| = 3$$

$$|\mathcal{C}(x) \cup \mathcal{C}(y)| = |\{x, y, 2, 3, 4, 5, 6, 7, 8\}| = 9, \text{ so}$$

$$P_{xy} = 3/9 = 0.33.$$

Andreu found that the core set-based definition of inter-node similarity worked well for unweighted graphs (Andreu 78). The basic reasoning behind this definition is illustrated in Figure 5.6. The definition may be profitably

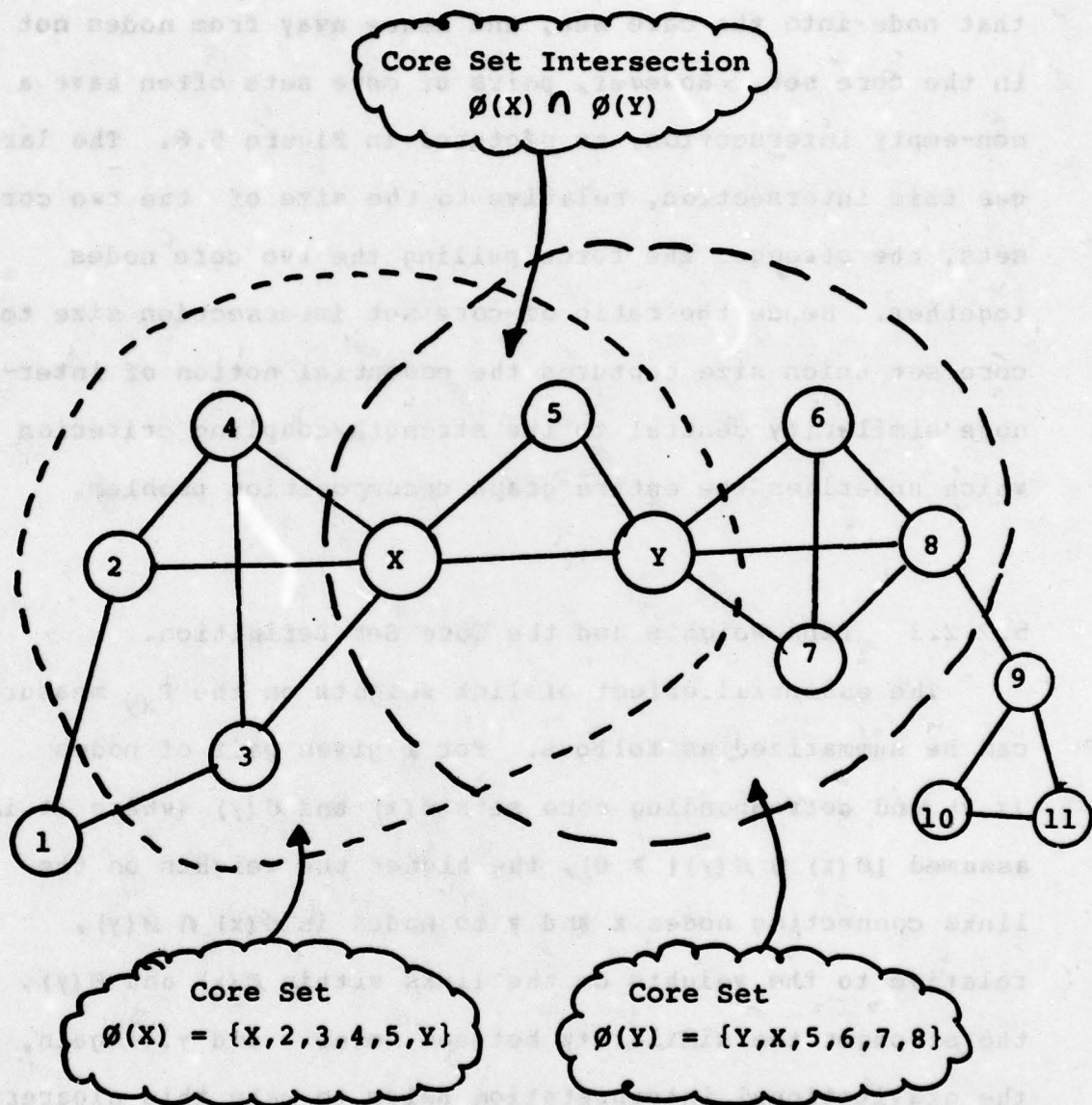


Figure 5.5

The core set concept.

viewed from a "gravitational" point of view: the larger a given node's core set, the stronger is the "force" pulling that node into the core set, and hence away from nodes not in the core set. However, pairs of core sets often have a non-empty intersection, as pictured in Figure 5.6. The larger this intersection, relative to the size of the two core sets, the stronger the force pulling the two core nodes together. Hence the ratio of core set intersection size to core set union size captures the essential notion of inter-node similarity central to the strength/coupling criterion which underlies the entire graph decomposition problem.

5.2.2.3 Link Weights and the Core Set Definition.

The essential effect of link weights on the P_{xy} measure can be summarized as follows. For a given pair of nodes $\{x, y\}$ and corresponding core sets $\mathcal{C}(x)$ and $\mathcal{C}(y)$ (where it is assumed $|\mathcal{C}(x) \cap \mathcal{C}(y)| > 0$), the higher the weights on the links connecting nodes x and y to nodes in $\mathcal{C}(x) \cap \mathcal{C}(y)$, relative to the weights on the links within $\mathcal{C}(x)$ and $\mathcal{C}(y)$, the stronger the similarity between nodes x and y . Again, the gravitational interpretation helps to make this clearer. Link weights may be viewed as "moderators" of the gravitational effect of $\mathcal{C}(x)$, $\mathcal{C}(y)$, and $\mathcal{C}(x) \cap \mathcal{C}(y)$ upon x and y . That is the link weight on links from x to $\mathcal{C}(x) \cap \mathcal{C}(y)$ and

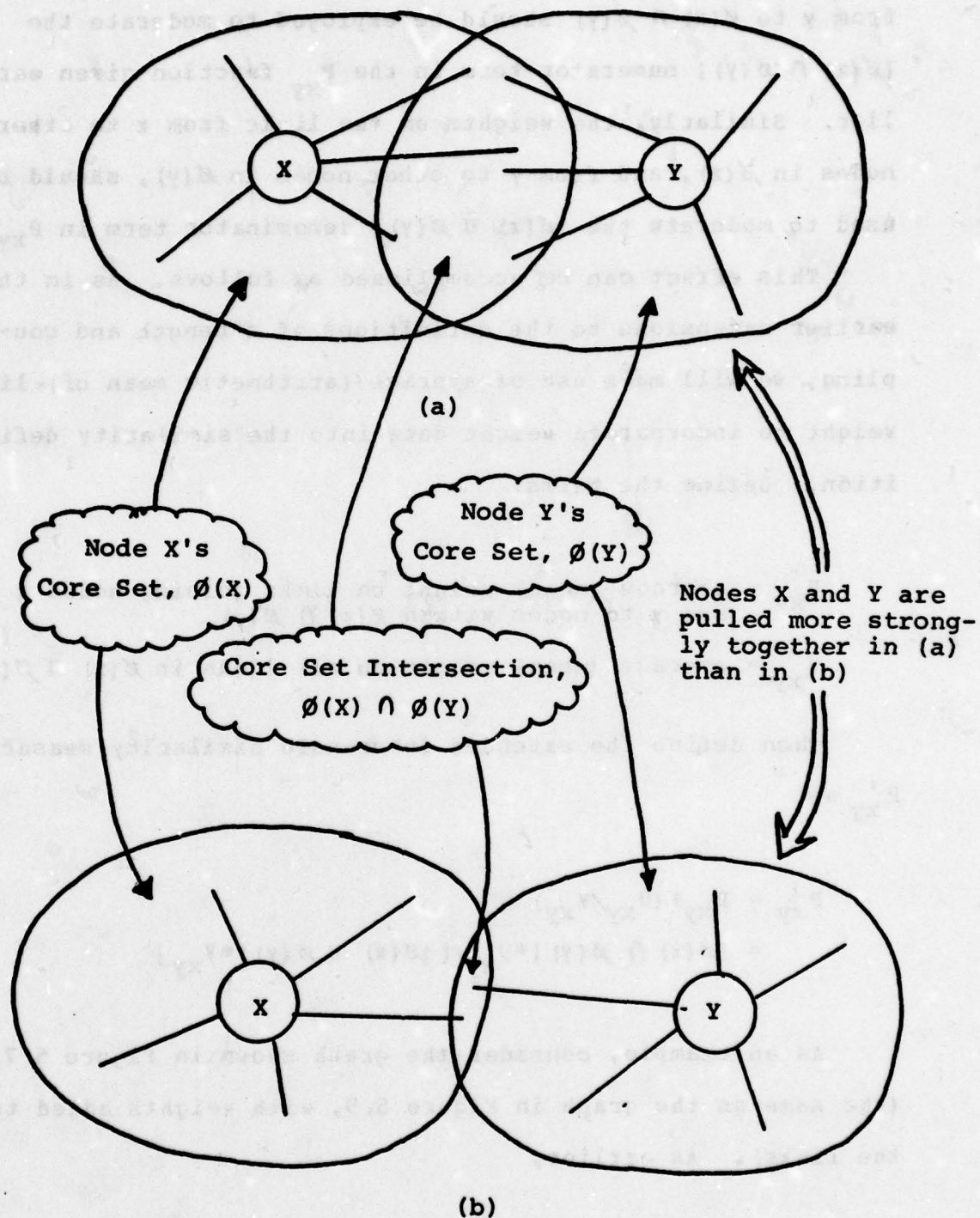


Figure 5.6

Core sets - the "gravitational" interpretation

from y to $\mathcal{N}(x) \cap \mathcal{N}(y)$ should be employed to moderate the $|\mathcal{N}(x) \cap \mathcal{N}(y)|$ numerator term in the P_{xy} function given earlier. Similarly, the weights on the links from x to other nodes in $\mathcal{N}(x)$, and from y to other nodes in $\mathcal{N}(y)$, should be used to moderate the $|\mathcal{N}(x) \cup \mathcal{N}(y)|$ denominator term in P_{xy} .

This effect can be accomplished as follows. As in the earlier extensions to the definitions of strength and coupling, we will make use of average (arithmetic mean of) link weight to incorporate weight data into the similarity definition. Define the terms:

U_{xy} = average (mean) weight on links joining nodes x and y to nodes within $\mathcal{N}(x) \cap \mathcal{N}(y)$

V_{xy} = average (mean) weight on all links in $\mathcal{N}(x) \cup \mathcal{N}(y)$

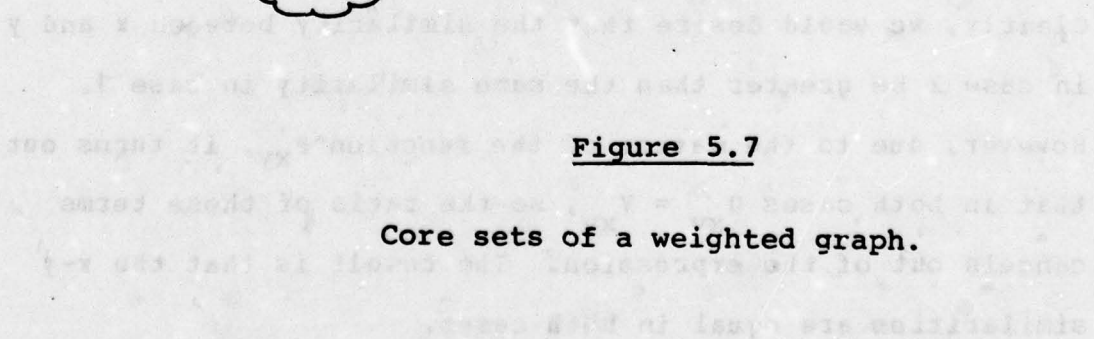
Then define the extended inter-node similarity measure P'_{xy} as

$$\begin{aligned} P'_{xy} &= P_{xy} * (U_{xy} / V_{xy}) \\ &= |\mathcal{N}(x) \cap \mathcal{N}(y)| * U_{xy} / [|\mathcal{N}(x) \cup \mathcal{N}(y)| * V_{xy}] \end{aligned}$$

As an example, consider the graph shown in Figure 5.7 (the same as the graph in Figure 5.5, with weights added to the links). As earlier,

$$\mathcal{N}(x) = \{2, 3, 4, 5, y\}$$

$$\mathcal{N}(y) = \{x, 5, 6, 7, 8\}$$



Core sets of a weighted graph.

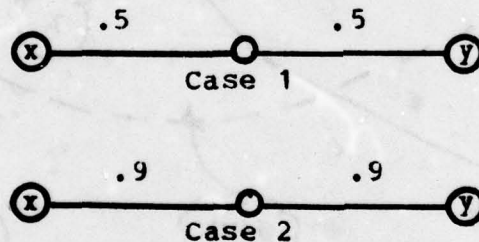
$$U_{xy} = (0.5+0.4+0.7)/3 = 0.533$$

$$V_{xy} = [(.6+.4+.3+.5+.4) + (.4+.7+.7+.6+.4)] / 10 \\ = .500 .$$

$$\text{So } P'_{xy} = (3/9) (.533/.500) = 0.356 .$$

5.2.2.4 A Further Adjustment.

A variety of test cases was examined and the measure P'_{xy} behaved appropriately in almost all cases. There turned out to be one important instance in which P'_{xy} did not produce the results that were to be expected, however. The nature of this problem may be easily illustrated. Consider the two 3-node weighted graphs shown below.



Clearly, we would desire that the similarity between x and y in case 2 be greater than the same similarity in case 1. However, due to the nature of the function P'_{xy} , it turns out that in both cases $U_{xy} = V_{xy}$, so the ratio of these terms cancels out of the expression. The result is that the x - y similarities are equal in both cases.

In general, this cancellation effect will occur whenever the following condition holds:

$$\phi(x) \cap \phi(y) = \phi(x) \cup \phi(y) - \{x, y\}.$$

That is, whenever the core set union and intersection differ only by the nodes x and y themselves. This is not at all an unusual occurrence, hence an additional modification must be made to P_{xy} in order to override this cancellation effect.

A good deal of experimentation with a number of possible adjustments led to the following simple change: replace the term U_{xy} with U_{xy}^2 in the P'_{xy} definition. That is, define

$$\begin{aligned} P_{xy} &= (U_{xy}) P'_{xy} \\ &= |\phi(x) \cap \phi(y)| * U_{xy}^2 / [|\phi(x) \cup \phi(y)| * v_{xy}] \end{aligned}$$

This simple change really has the effect of scaling all the P_{xy} values by the factor U_{xy} . In the special case described above, it insures that the link weights on the x - y path do have an impact, as desired.

To see this impact, consider the two 3-node graphs discussed a moment ago. Now we would find that

$$\text{Case 1 : } P_{xy} = (1/3) (0.5 / 0.5) = .167$$

$$\text{Case 2 : } P_{xy} = (1/3) (0.9 / 0.9) = 0.300,$$

a much more reasonable result than that obtained earlier.

5.2.2.5 Some Test Cases Using P_{xy}

In order to better see the operational effect of the similarity function P_{xy} , it is worthwhile examining sequences of simple graphs, in which a single factor is changed from one step to the next. Figures 5.8 and 5.9 contain some interesting sequences. In Figure 5.8, in the sequence a-b-c-d, more and more nodes are added to $O(x)$ and $O(y)$ individually, but these nodes do not impact the intersection. All link weights are kept constant at 0.5. The effect is to "pull apart" nodes x and y , i.e., to make it increasingly desirable that two clusters should be created by severing the x - y link. As is shown, the similarity P_{xy} correspondingly decreases through the sequence, as is desired.

Now in the sequence b-e-f-g, the graph structure is held fixed while link weights are altered. In (b), $P_{xy} = 0.25$. In (e), the link weight w_{xy} is increased from .5 to .9, while the other two weights are decreased to .2. Clearly, we would expect the similarity P_{xy} to increase, and it does (to 0.93). In (f) the opposite changes to link weights result in P_{xy} decreasing (as expected) to 0.03.

Finally, in the sequence c-g-h-i-j, similar intuitively correct results are also seen. In particular, if (i) is compared to (f), it is seen that the impact on P_{xy} is slightly greater in the former case, when link weights are

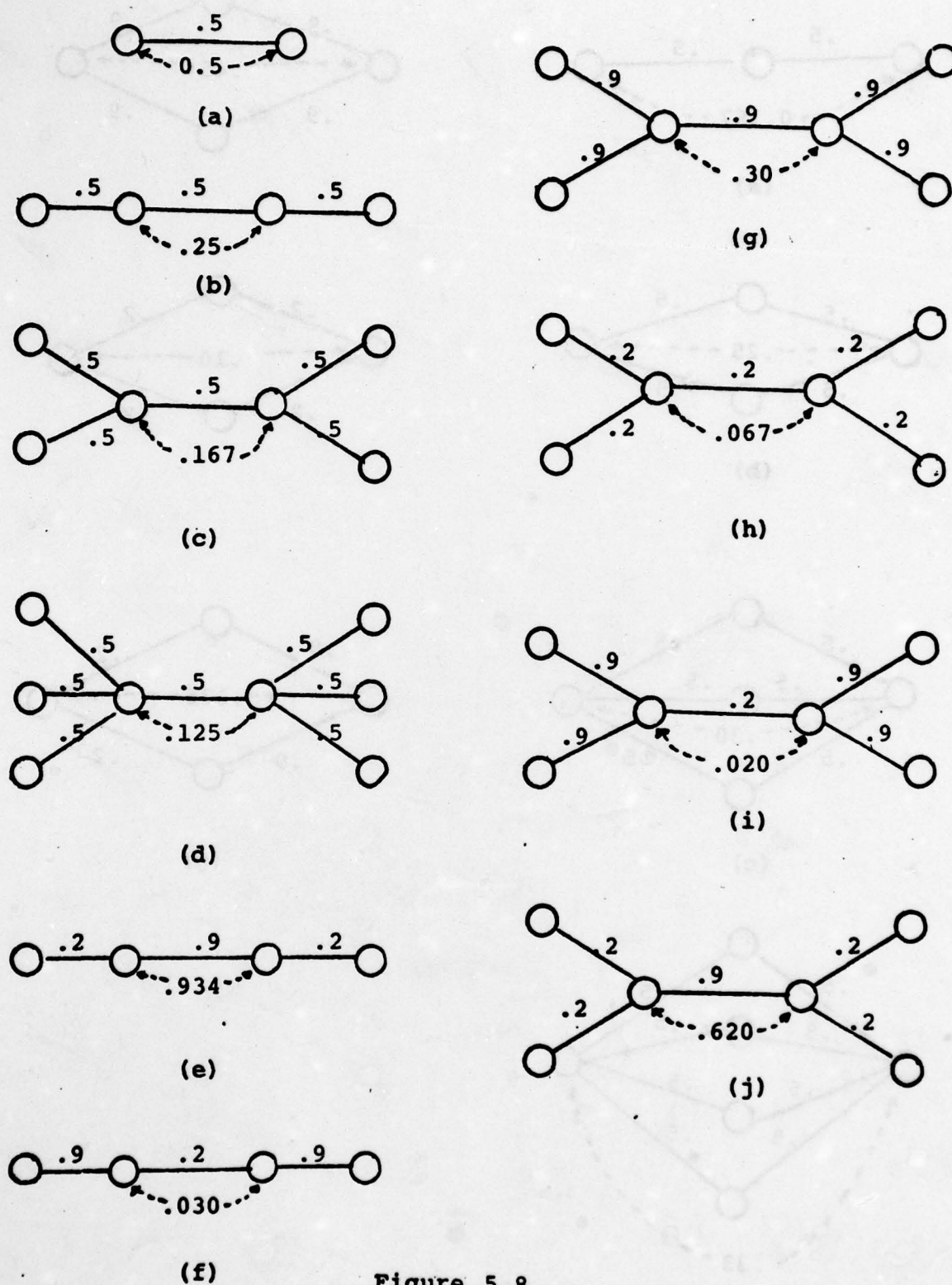
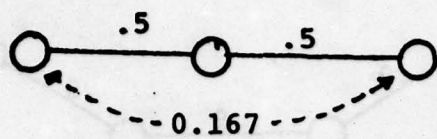
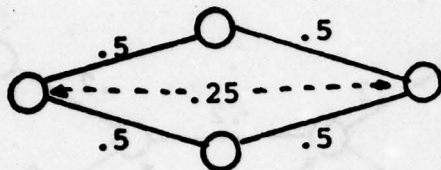
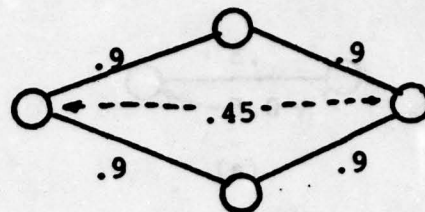


Figure 5.8

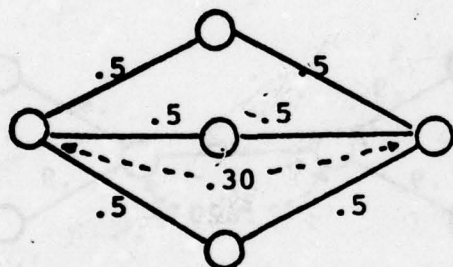
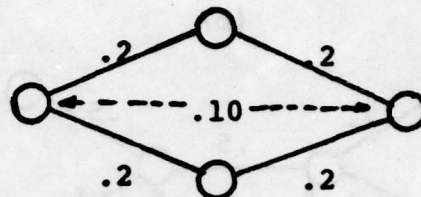
Examples illustrating behavior of similarity measure.



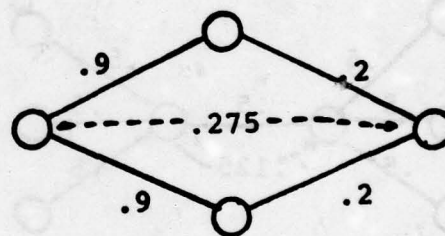
(a)



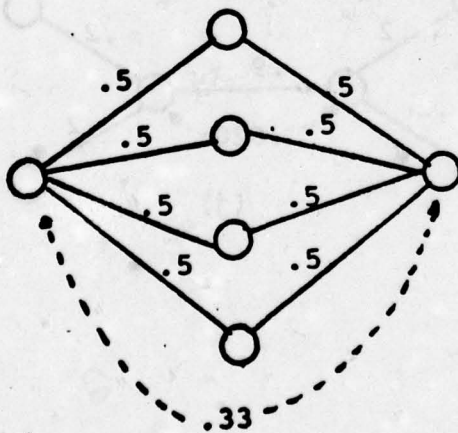
(b)



(c)



(g)



(d)

Figure 5.9

Examples illustrating behavior of
similarity measure.

changed from all 0.5 to the 0.9-0.2-0.9 pattern. This is reasonable, since the extra nodes in (i) would be expected to exert an even greater "gravitational" effect on x and y than in (f) as a result of the link weight change.

The patterns exhibited in Figure 5.9 are equivalently intuition-supporting. It should be noted that the graphs in Figure 5.9 all have the property that

$$\delta(x) \cap \delta(y) = \delta(x) \cup \delta(y) - \{x, y\}.$$

Hence if the measure P'_{xy} were used, the sequence b-e-f-g would all exhibit precisely the same similarity between nodes x and y. Obviously this result would not conform to intuition: in (e), for instance, x and y would be expected to be more similar than in (f). The use of P_{xy} instead of P'_{xy} insures that this is the case.

5.3 CLUSTERING ANALYSIS TECHNIQUES USING THE EXTENDED MODEL.

The main purpose of cluster analysis is to "group similar objects" (Hartigan 75). While there is a large number of individual techniques available in the clustering literature, they may be broadly categorized into two groups: agglomerative (or "bottom-up") techniques and partitioning ("top-down") techniques. The former class of techniques begins with each point (node) being viewed as a separate cluster, then proceeds to join together the "most similar" pair of clusters. The merging process is repeated until a single cluster remains.

Partitioning techniques move in the opposite direction. Beginning with a single encompassing cluster, they proceed to break up the cluster into two (or more) sub-clusters. After each cycle, a decision must be made as to which current cluster should be partitioned next.

There are also other "hybrid" techniques that possess aspects of both classes. "Leader" techniques, for example, begin by partitioning the entire set into a set of especially strongly connected clusters ("leader" clusters) plus unallocated elements. Special methods are then used to decide what to do with the unallocated points - i.e., assign them to one of the leader clusters, or group some of them together to form additional clusters

A discussion of clustering techniques used in the earlier SDM analysis is given in (Andreu 78). In general, the most effective techniques (in terms of both algorithm execution speed and ability to locate good decompositions) have been basic bottom-up clustering approaches (to be described shortly). Andreu experimented with some other approaches but found them to be too inefficient in terms of solution time to be used on graphs of nontrivial size. A new top-down partitioning technique that exhibits good execution speed has recently been developed by this author, and is compared in performance against the various clustering algorithms in Chapter 6. Many other techniques exist that have yet to be explored, and a few of the most promising ones are briefly discussed in a later section of this chapter.

5.3.1 Four Hierarchical Clustering Techniques.

At this point four different hierarchical clustering techniques that have been used most frequently in SDM analysis, and which are presently included in the SDM PL/1 analysis package, will be described. All four techniques are based on similarity (as opposed to distance) coefficients. Figure 5.10 illustrates three clusters of graph nodes, shown for simplicity as points. (Once a similarity matrix has been computed, the information originally conveyed by the graph

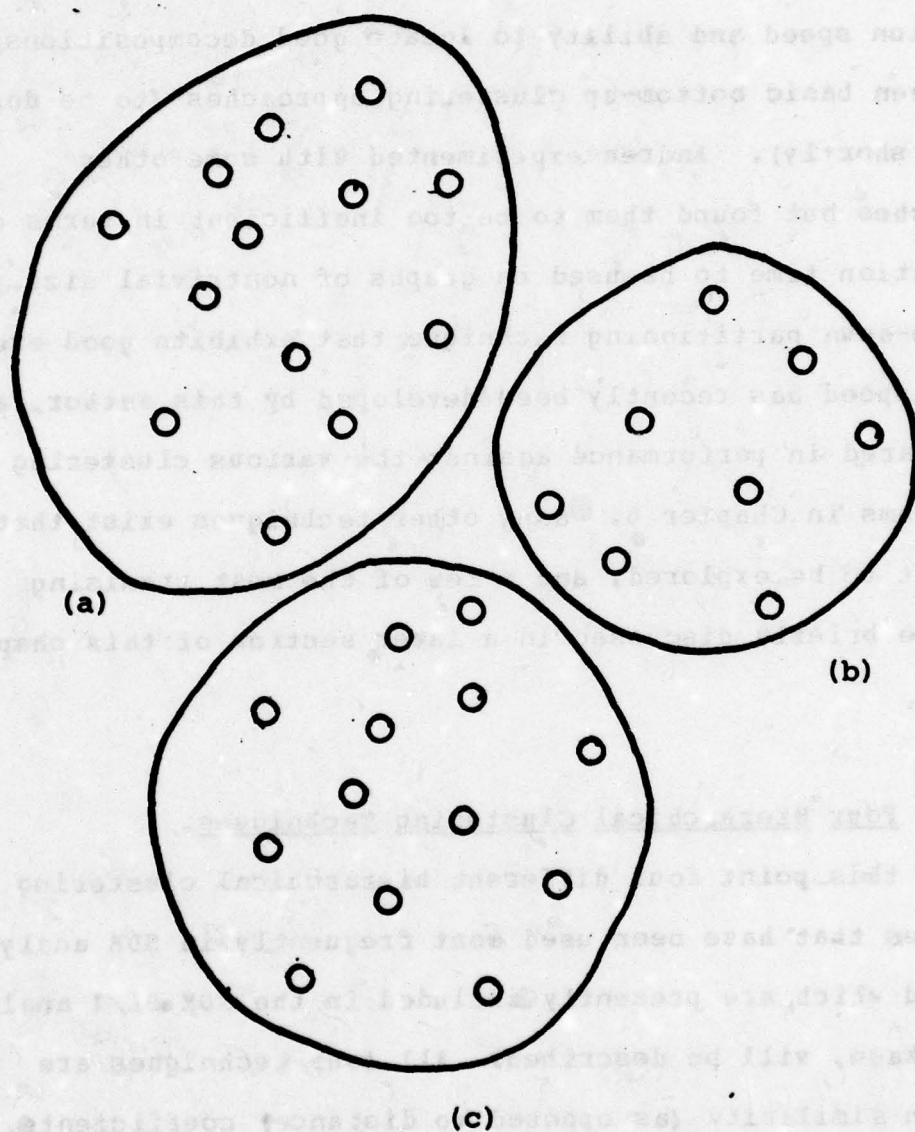


Figure 5.10

Three subsets, ready for the next "merge" decision.

links and weights has been absorbed into the similarity values, hence the actual graph structure loses importance.) Assume that the original unclustered points (nodes) have been partially clustered to the stage indicated in Figure 5.10. The next clustering decision is the determination of the best pair of clusters, (a,b), (a,c), or (b,c), to be merged at the next step.

For each of the methods (termed HIER1, HIER2, HIER3, and HIER4) a criterion value is calculated for each cluster pair that may be merged at that step. The cluster pair with the largest criterion value is then merged to form a single cluster, producing the next level up in the clustering hierarchy. This process is then repeated until a single cluster remains.

5.3.1.1 Single Linkage Clustering (HIER1).

The essence of the clustering decision involves the question of what is meant by the closeness between two sets of points, given that the closeness between each pair of points is quantitatively known. Perhaps the simplest interpretation of set closeness is that employed in the "single linkage" clustering algorithm: the closeness between two sets A and B is the closeness between the closest pair of points (a,b) such that $a \in A$ and $b \in B$. The algorithm derives its name from the fact that only a single pair of

points (a,b) need be especially "close" in order that the entire sets A and B be judged to be "close". While the single linkage algorithm generally gives good results, it can sometimes lead to unusual clustering patterns, notably the "strung out" pattern illustrated in Figure 5.11.

Single linkage clustering, then, is formalized accordingly:

For each pair of clusters (X,Y), calculate

$$P_{XY} = \{ \max_{\substack{i \in X \\ j \in Y}} (p_{ij}) \},$$

where p_{ij} = similarity between points i and j.

Then merge the clusters (X*,Y*) such that

$$P_{X^*Y^*} \geq P_{XY} \quad \forall \quad X,Y.$$

5.3.1.2 Complete Linkage Clustering (HIER2).

Single linkage may be viewed as a "risk-prone" algorithm: the algorithm is willing to presume that, if the point pair (a,b) is close, the other points in A and B will be close also, hence A and B should be merged. In contrast, complete linkage is a "risk-averse" algorithm. Rather than make the assumption stated above, this algorithm insures it by seeking to merge the cluster pair such that the least similar points are closest. That is, under maximum linkage, the pair of clusters to be merged at any stage is determined by:

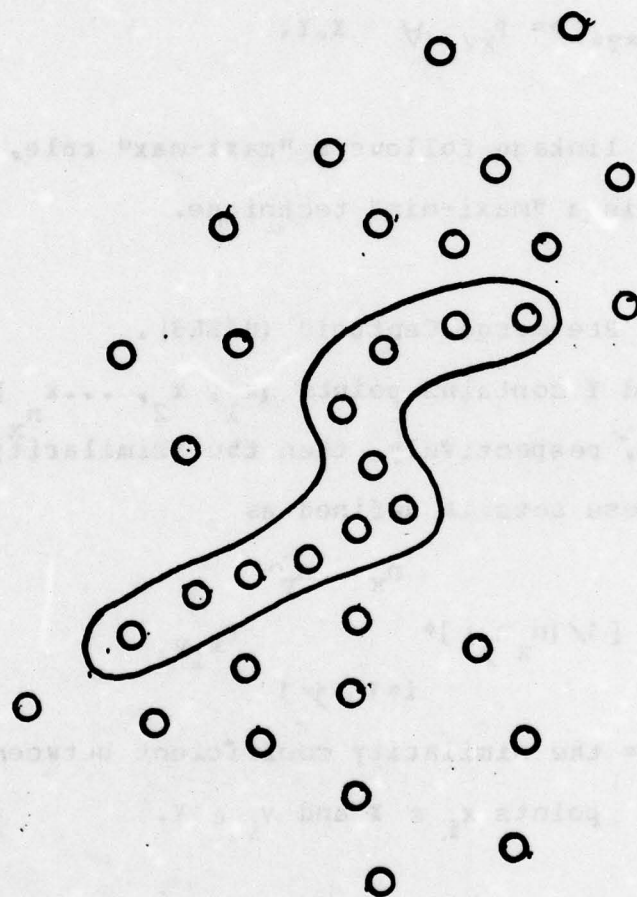


Figure 5.11

Single-linkage clustering anomaly.

For each pair of clusters (X, Y) , calculate

$$P_{XY} = \{ \min_{\substack{i \in X \\ j \in Y}} (p_{ij}) \}$$

Then merge the cluster pair (X^*, Y^*)

such that $P_{X^*Y^*} \geq P_{XY} \quad \forall \quad X, Y.$

Thus, single linkage follows a "maxi-max" rule, while complete linkage is a "maxi-min" technique.

5.3.1.3 Largest Pre-merge Centroid (HIER3).

If sets X and Y contains points $\{x_1, x_2, \dots, x_{n_x}\}$, $\{y_1, y_2, \dots, y_{n_y}\}$, respectively, then the "similarity centroid" between these sets is defined as

$$D_{xy} = \left[\frac{1}{(n_x n_y)} \right] * \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} p_{x_i y_j}$$

where $p_{x_i y_j}$ = the similarity coefficient between points $x_i \in X$ and $y_j \in Y$.

D_{XY} is a measure of "average similarity" between the sets X and Y .

The largest pre-merge centroid algorithm makes use of the D_{XY} measure: at any stage in the clustering, the cluster pair (X, Y) with the largest value of D_{XY} is selected for the next merge. Formally:

For each pair of clusters (X,Y) , calculate D_{XY} ,

then merge cluster pair (X^*,Y^*) such that

$$D_{X^*Y^*} \geq D_{XY} \quad \forall \quad X,Y.$$

5.3.1.4 Largest Post-Merge Centroid (HIER4).

If set X contains points $\{x_1, x_2, \dots, x_n\}$ then the "similarity centroid" of the set X is defined as:

$$D_X = \left[1 / ((n_X)(n_X - 1) / 2) \right] \sum_{i=1}^{n_X-1} \sum_{j=i+1}^{n_X} P_{x_i x_j}$$

where $P_{x_i x_j}$ = the similarity measure between points $x_i \in X$ and $x_j \in X$.

D_X is a measure of the "internal similarity" of the set X . The measure D_X is used in the largest post-merge similarity algorithm: the cluster pair (X,Y) such that, when merged, exhibits the largest internal similarity value, is selected as the next pair for merging. Formally, we have,

For each pair of clusters (X,Y) , let $Z \leftarrow (X,Y)$

and calculate D_Z . Then merge the cluster pair

(X^*,Y^*) such that $D_{Z^*} \geq D_Z \quad \forall \quad Z$.

5.3.2 Comparative Analysis of Clustering Methods.

Each of the clustering algorithms described in the previous section makes good intuitive sense. There is no obvious a priori way of choosing among them - i.e., of determining which one would tend to produce the best results in a typical SDM graph analysis. For this reason, all four algorithms are included in the SDM analysis package, and a user of the package may apply whichever one he chooses, or all four.

However, it is worthwhile to explore somewhat the question of dominance: does one (or more) of the algorithms tend to produce consistently superior decompositions relative to the others? A related question concerns efficiency: are certain of the clustering algorithms significantly faster executing than others in general? Of course, these questions cannot be answered for all possible cases. However, the experiment reported here will at least give some clues.

For this experiment, a "random graph generator" was developed, a series of graphs generated, and each clustering method applied to each graph. The results are discussed below.

The random graph generator (written in PRIME 400 extended BASIC) functions as follows. The user is requested to supply the node count for the graph to be generated, as

well as the mean (M_g) and variance (V_g) of the distribution of links-per-node for the target graph. In developing the generator, it was assumed that the number of links per node would follow a normal distribution.

The generator, for each graph node, then proceeds to select normal random numbers from $N(M_g, V_g)$ (truncated at 0) to use in allocating links to each node. To determine the "recipient" node for each such link, the generator again draws a random number, from the uniform distribution $U[1, \text{nodecount}]$. Finally, the link weight is likewise randomly selected, from $U[0, 1]$, modified so as to produce the values 0.1, 0.2, ..., 0.9 with equal likelihood.

A series of 7 graphs was generated using the random graph generator, to be used in the comparative analysis. Furthermore, it was felt that "random" graphs, while providing unbiased cases, probably do not exhibit as much structure as would real-world cases. Therefore, a set of six non-random graphs, generated by hand to exhibit significant clustering structure, was also included in the analysis. The specifications of the various test graphs are given in Table 5.1.

The results of the experiment are given in Table 5.2. This exhibit shows the measure for the best achieved decomposition for each of the four decomposition methods. A summary of the information from Table 5.2 is contained in

	Graph ID Number	Number of Nodes	Mean Links per Node	Variance of LPN	Total No. of links
randomly gener- ated	1	40	2.5	1.0	61
	2	22	3.5	1.5	42
	3	22	3.5	1.5	37
	4	15	3.5	1.5	27
	5	15	4.0	1.2	31
	6	15	2.5	1.2	24
	7	10	3.5	1.5	14
non- random	8	22	3.3	1.6	37
	9	15	2.9	0.9	22
	10	10	2.6	0.5	13
	11	6	2.3	0.5	7
	12	6	2.3	0.5	7
	13	6	2.3	0.5	7

Table 5.1

Specifications for random and non-random graphs.

Graph ID Number	Number of Nodes	M value for best located decomposition (number of clusters in best result)			
		HIER1	HIER2	HIER3	HIER4
randomly generated graphs	1	.076 (5)	.098 (5)	.036 (3)	.021 (2)
	2	.04 (1)	.04 (1)	.10 (2)	.07 (3)
	3	.035 (3)	.046 (5)	.079 (4)	.080 (3)
	4	.24 (3)	.24 (3)	.24 (3)	.15 (4)
	5	.08 (1)	.11 (2)	.08 (1)	.08 (1)
	6	.052 (1)	.052 (1)	.052 (1)	.061 (2)
	7	.049 (2)	.056 (3)	.056 (3)	.085 (2)
non- random	8	.25 (3)	.29 (3)	.41 (4)	.41 (4)
	9	.39 (3)	.27 (2)	.39 (3)	.26 (3)
	10	.48 (3)	.48 (3)	.48 (3)	.48 (3)
	11	.28 (2)	.28 (2)	.28 (2)	.28 (2)
	12	.05 (1)	.05 (1)	.05 (1)	.05 (1)
	13	.06 (1)	.06 (1)	.06 (1)	.19 (2)

Table 5.2

Relative performance of the four hierarchical clustering techniques

	HIER1	HIER2	HIER3	HIER4
Clear winner	0	2	1	4
Tied for best	5	4	6	4
2nd or tied for 2nd	4	5	5	3
3rd or tied for 3rd	3	2	1	1
4th or tied for 4th	1	0	0	1

Table 5.3

Relative performance for the clustering routines.

<u>Weight</u>	<u>Category</u>
4	Clear Winner
3	Tied for 1st
2	2nd or tied for 2nd
1	3rd or tied for 3rd
0	4th or ties for 4th

<u>Algorithm</u>	<u>Ranking</u>
HIER1	26
HIER2	32
HIER3	33
HIER4	35

Table 5.4

Ranking for clustering routines.

Table 5.3. There it may be seen that all the algorithms except HIER1 (single linkage) produced a "clear winner" in at least one test case (in particular, in at least one randomly generated test case). HIER1, even though not producing a clear winner, did show the second-highest rate of producing "ties for best." The net result is that the first, third, and fourth algorithms somewhat outperform the first, although all four algorithms perform the decomposition task reasonably effectively.

A simple effectiveness ranking may be produced by arbitrarily assigning a 4 for producing a "clear winner", 3 for "tied for best", etc. The ranking that results is given in Table 5.4. This ranking reiterates the fact that three of the four algorithms (HIER2, HIER3, and HIER4) are essentially equivalently powerful in determining good decompositions, while the fourth (HIER1) seems somewhat less effective.

The conclusion to be gained from this effectiveness test is that it is useful (and important) to have a variety of algorithms at hand to bring to bear on such decomposition tasks. There is a nontrivial risk that, in any given problem, any one algorithm will produce a considerably inferior graph decomposition.

All the clustering algorithms are reasonably fast in terms of computer execution time. Rough measurements

obtained during the foregoing tests indicate that HIER1 is somewhat (e.g., 40 percent) faster than HIER2, HIER3, and HIER4. The latter three all seem to execute with roughly equal speed. As a benchmark, each of the latter three algorithms required approximately 3 CPU seconds on a 370/168 to perform complete clustering on the 40-node graph used in the comparative analysis. All the clustering algorithms are bounded by an execution speed proportional to n^2 .

5.3.3 A "Greedy" Clustering Algorithm.

A somewhat different approach to clustering, motivated by the work of Ward (Ward 63), was also investigated. Ward suggested that a general approach to clustering might be based on the notion of seeking to optimize some objective function to be specified by the investigator. Ward's own approach was to maximize the mean squared error function; that is, at each clustering step, the cluster pair to be merged is the pair that leads to the minimum increase (or maximum decrease) in the within-group mean squared error. Ward's particular criterion is only applicable to clustering problems wherein corresponding to each point is a vector of data values (e.g., the points might be individuals, the data values might be height, weight, etc.).

Ward's general objective function maximization approach may be applied to the SDM graph clustering problem, by

adopting a different objective function. The most appropriate candidate for objective function is the decomposition goodness measure, M . Following this criterion, at any stage in the clustering, the cluster pair (p,q) would be merged was greater than for all other potential cluster mergers. More precisely:

At stage k , there are t clusters,

$$\begin{aligned} & \{C_{k_1}, C_{k_2}, \dots, C_{k_t}\} \\ \text{Let } D_k &= \{C_{k_1}, C_{k_2}, \dots, C_{k_t}\} \\ D_k(i,j) &= \{C_{k_1}, C_{k_2}, \dots, \{C_{k_i} \cup C_{k_j}\}, \dots, C_{k_t}\} \\ &= \{D_k - C_{k_i} - C_{k_j}\} \cup \{C_{k_i} \cup C_{k_j}\} \\ &= \text{the new decomposition obtained from } D_k \end{aligned}$$

by merging clusters K_i and K_j .

$$\text{Define } \Delta M_k(i,j) = M(D_k(i,j)) - M(D_k).$$

Then merge clusters K_{i^*} and K_{j^*} such that

$$\Delta M_k(i^*, j^*) = \max_{i,j} \{\Delta M_k(i,j)\}.$$

Since this approach to clustering follows a path of local steepest ascent, termed a "greedy" algorithm: the algorithm tries to "get all it can," for the given objective function, at each step. Of course, such an algorithm is also "myopic," in that it only concerns itself with the best move at each stage; large increases early in the clustering may lead to poor results later on. It is not clear at this

point whether such a greedy algorithm would do as well as the more conventional techniques discussed earlier.

In order to study the effectiveness of the greedy approach, this algorithm was programmed in PL/1 and added to the interactive analysis package. Certain of the graphs used in the comparative analysis reported in the previous section were also decomposed using the greedy algorithm.

First of all, the "greedy" algorithm in its present form is essentially inapplicable, as it is simply too inefficient. Decomposing a 10-node graph, for example, requires on the order of 15 370/168 CPU seconds. This is not to say that this algorithm need necessarily be hopelessly slow, however, as discussed below.

Putting aside efficiency considerations momentarily, the "greedy" algorithm seems to perform in an unusual manner. In two of the test cases studied (cases 6 and 7), this algorithm produced the same decomposition as the best of the other four techniques. However, in two other cases (cases 10 and 11), "greedy" was unable to find the best decomposition, even though all four of the other algorithms did find it. While the cases studied here are limited because of the above mentioned efficiency problems, "greedy's" performance seems decidedly mixed.

The efficiency problem with "greedy" stems from the fact that a large number goodness) must be made, especially

during the earlier clustering stages. In its present form, these M-calculations are carried out in their entirety (i.e., no approximations are introduced). While any given M-calculation is not terribly time-consuming (requiring perhaps .02 CPU seconds), the large number of such calculations made by "greedy" rapidly add up. For example, in the first clustering stage alone, for a simple 15-node graph, there would be

$$(14 + 13 + 12 + \dots + 1) = 7(15) = 105$$

such calculations, requiring approximately two CPU seconds.

The "greedy" algorithm could probably be made acceptably efficient by developing an approximation to the M criterion that is employed within it. Such an approximation, potentially suitable to this algorithm, has been developed on behalf of yet another graph decomposition technique entirely, the interchange algorithm (discussed in detail in the next chapter). However, in light of the mixed performance of "greedy" in the early studies, as its applicability and use within the "greedy" algorithm has not yet been explored.

5.3.4 Other Approaches to Graph Decomposition.

In this section, a variety of other approaches to the decomposition of weighted graphs will be identified and briefly discussed. While the techniques to be presented

here (with one exception) have not been incorporated into the SDM analysis package, they are all potentially appropriate for that purpose, pending further testing. Future extensions to the package might include one or more of these techniques.

5.3.4.1 A Leader Technique.

The basic idea underlying leader (sometimes called "core") techniques is to isolate a few non-overlapping, strongly coherent subgraphs, then to perform additional analysis to determine what to do with the leftover nodes (if any).

Following the thinking underlying the goodness measure M , we would like to identify leader subgraphs that have especially high strength, and which are relatively weakly coupled to the other subgraph nodes (notably, to nodes in the other leader subgraphs).

Andreu derived such an algorithm for identifying leader subgraphs, described in (Andreu 78, pages 124-133).

Andreu's algorithm is based on the notion of node connectivity. The connectivity of node i is simply the number of nodes that are linked to node i (recall Andreu worked with binary links). The algorithm then

1. isolates a subset, U , of nodes with highest connectivity;

2. finds the node of this subset with the largest "kernel subset," where the kernel subset of node i is the set of all nodes that are members of CS (the core set of node i) but not members of the core sets of other nodes in the subset U ;
3. isolates that kernel subset as a leader subgraph, and reduces the original graph accordingly;
4. repeats the first three steps for the remaining nodes in U until of the possible stopping conditions is reached.

This leader subgraph technique could be extended to analyze graphs with weighted links by re-defining the concept of connectivity. The following definition of c , the connectivity of node i , would serve to identify those nodes that are both thickly (many links) and strongly (high link weights) connected in the network:

Let CS_i = core set of node i

Then c_i = connectivity of node i

$$= \sum_{\substack{j \in CS \\ j \neq i}} a_{ij}$$

That is, c_i is simply the sum of the weights on the links within CS_i .

Re-defining c_i in this fashion, then applying the remaining steps of Andreu's algorithm, will tend to isolate leader subgraphs which are both thickly and strongly con-

nected within themselves. Both characteristics are important for good leader subgraphs, as it is equally important, in decomposing the requirements graph, to avoid cutting a large number of links as it is to avoid cutting links with high weights.

One potential difficulty with the leader subgraph technique, one which Andreu does not really address, concerns what to do with leftover nodes (i.e., nodes that do not become members of one of the kernel subsets). There are various approaches that may be taken - lump leftovers into the subgraph to which each is most strongly connected, group certain leftovers to form a new cluster, etc. - but specific implementation techniques may prove challenging.

5.3.4.2 The Bond Energy Approach.

A new cluster analysis algorithm was developed by McCormack (McCormack, et. al. 71), and has recently been applied effectively to certain cluster applications similar in nature to SDM decomposition (Hoffer & Severance 75).

This technique operates directly upon a graph adjacency matrix. By permuting the rows and columns of the matrix in such a way as to push numerically larger array elements together, this algorithm serves to identify the natural groups and clusters that occur in the data, as well as the associations of these groups with one another. The authors

have named their algorithm the "bond energy" technique. Central to it is a "measure of effectiveness", or ME, which is used to quantify the "clumpiness" of a given permutation of the rows and columns of the array. The ME is larger for an array which possesses dense clumps of numerically large elements as compared to an equivalent array in which the rows and columns have been permuted so that the large elements are more uniformly distributed.

McCormick suggests the following measure of effectiveness:

$$ME = \sum_{i=1}^N \sum_{j=1}^N a_{ij} (a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1})$$

Essentially, this ME is the sum over all node pair of the "bond energy" for each node pair, where bond energy for nodes (i,j) is calculated as

$$a_{ij} (a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1})$$

In matrix terms, the bond energy for node pair (i,j) is the product of the four nearest-neighbor adjacency values (weights) for that node pair.

The algorithm used to locate the row/column permutation with the highest ME involves reducing the overall total $(N!)^2$ permutations to $2N$ calculations by applying the near-

est-neighbor feature of the maximand. While the algorithm does not guarantee optimal ME, its use has shown it to produce very good results in general. This algorithm basically involves arbitrarily placing one column in a permuted position, then placing each remaining column in turn in the position that produces the greatest contribution to ME. In the general case, this procedure would have to be repeated on the matrix rows; however, in the case of a symmetric matrix (the present case), no such row permutations need be executed.

The bond energy algorithm (BEA) suffers from the disadvantage that it gives no hierarchical trace. That is, it produces a single best clustering, not a hierarchical sequence of clusters. Consequently, additional mechanisms would have to be added "on top" of the BEA itself to allow such exploratory marginal analysis. Furthermore, some of the clustering and partitioning algorithms discussed earlier might be effectively combined with the BEA to allow such marginal analysis to be performed easily.

The BEA was illustrated by McCormack in an example problem that bore much similarity to the usual SDM context. This suggests that it might be a particularly fruitful avenue for investigation for SDM analysis.

5.3.4.3 Node Tearing Techniques.

The central idea underlying this class of decomposition techniques is to locate small separating sets - i.e., sets of nodes with low cardinality such that their removal from the graph splits the graph into two unconnected subgraphs. In this sense, the network is said to be "torn" in half - hence, node tearing.

A new algorithm for node tearing was reported recently, by Sangiovanni-Vincentelli (Sangiovanni-Vincentelli, et. al. 77). Many of the basic concepts used in this technique are similar to those used by Andreu in the leader subgraph approach discussed earlier (core set, connectivity, etc.), and their algorithm has much the same flavor as that of Andreu's, although its objective is somewhat different.

In particular, the algorithm proposed by Sangiovanni-Vincentelli presumes a binary graph. As in the case of the leader algorithm, however, it appears that it is feasible to extend it to incorporate link weight information into its operation. The key decision point in the algorithm is the choice of the "next iterating node" (see the reference for details). Sangiovanni-Vincentelli suggests a "greedy" strategy for making this decision, namely, to choose the node that minimizes the connectivity of a certain subgraph. If this criterion were changed, to "minimize the sum of the weights on the links within that subgraph," the modified

algorithm should function properly and take account of link weights appropriately. In particular, in the case where link weights are all equal, this modified algorithm should lead to the same results as the original.

5.3.4.4 The Interchange Algorithm.

Another top-down hierarchical partitioning technique has been developed, by the present author, specifically for the SDM graph decomposition problem. This algorithm successively partitions the "current graph" into two subgraphs, using a criterion derived directly from the decomposition goodness measure, M . Its detailed operation, and examples of its use, are given in the next chapter, so it will not be further elaborated upon here.

5.4 A CASE STUDY USING INTERDEPENDENCY WEIGHT EXTENSIONS.

In order to illustrate the application of the various techniques discussed in this paper, including the use of the SDM analysis package, we present here a decomposition analysis of a particular small design problem. The problem addressed is one that has been used in this research effort in the past: a set of 22 requirements and interdependencies for the design of a database management system. The requirements were originally developed as a simple test vehicle in an early phase of the SDM research (see Andreu 78), and have been referred to on a number of occasions. In particular, we made use of the 22-node system in illustrating various potential extensions to the SDM representational model in the previous chapter.

Source statements of the system's requirements, and the interdependency relationships, are given in the foregoing reference. Figure 5.12 shows the graphical representation for this design problem. The interdependency weights are given in coded terms, as discussed earlier: W means "weak," A, "average," and S "strong." A similar scheme is used to label strengths of interdependency relationships.

In the earlier analysis of this 22-node design problem, the basic (binary) graph model was used. An analysis of that graph produced the following best decomposition:

<u>CLUSTER</u>	<u>NODES INCLUDED</u>
----------------	-----------------------

1	1,2,3,5
2	6,9,21
3	4,16,17,18,22
4	8,10,11,12,19,20
5	7,13,14,15 .

This decomposition is illustrated in Figure 5.13.

5.4.1 Results from the Case Study.

Under the extended model and associated analytical techniques, a somewhat different clustering results. The results produced by each of the hierarchical clustering methods, and by the interchange partitioning technique, are given below.

<u>METHOD</u>	<u>BEST M</u>	<u>NUMBER OF CLUSTERS</u>
HIER1	0.04	1
HIER2	0.28	3
HIER3	0.33	3
HIER4	0.23	3
INTERCH	0.38	4

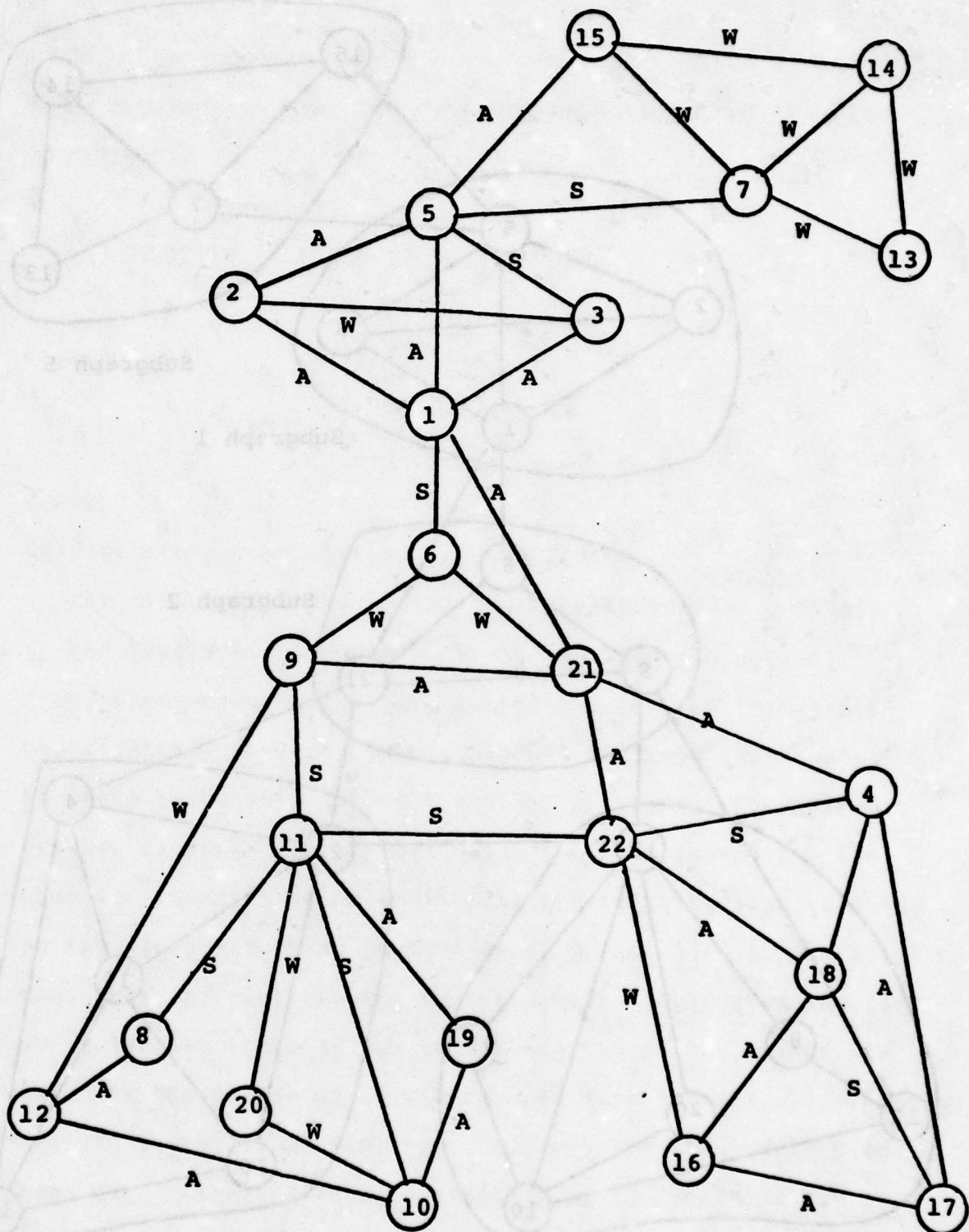


Figure 5.12

The 22-node DBMS weighted requirements graph

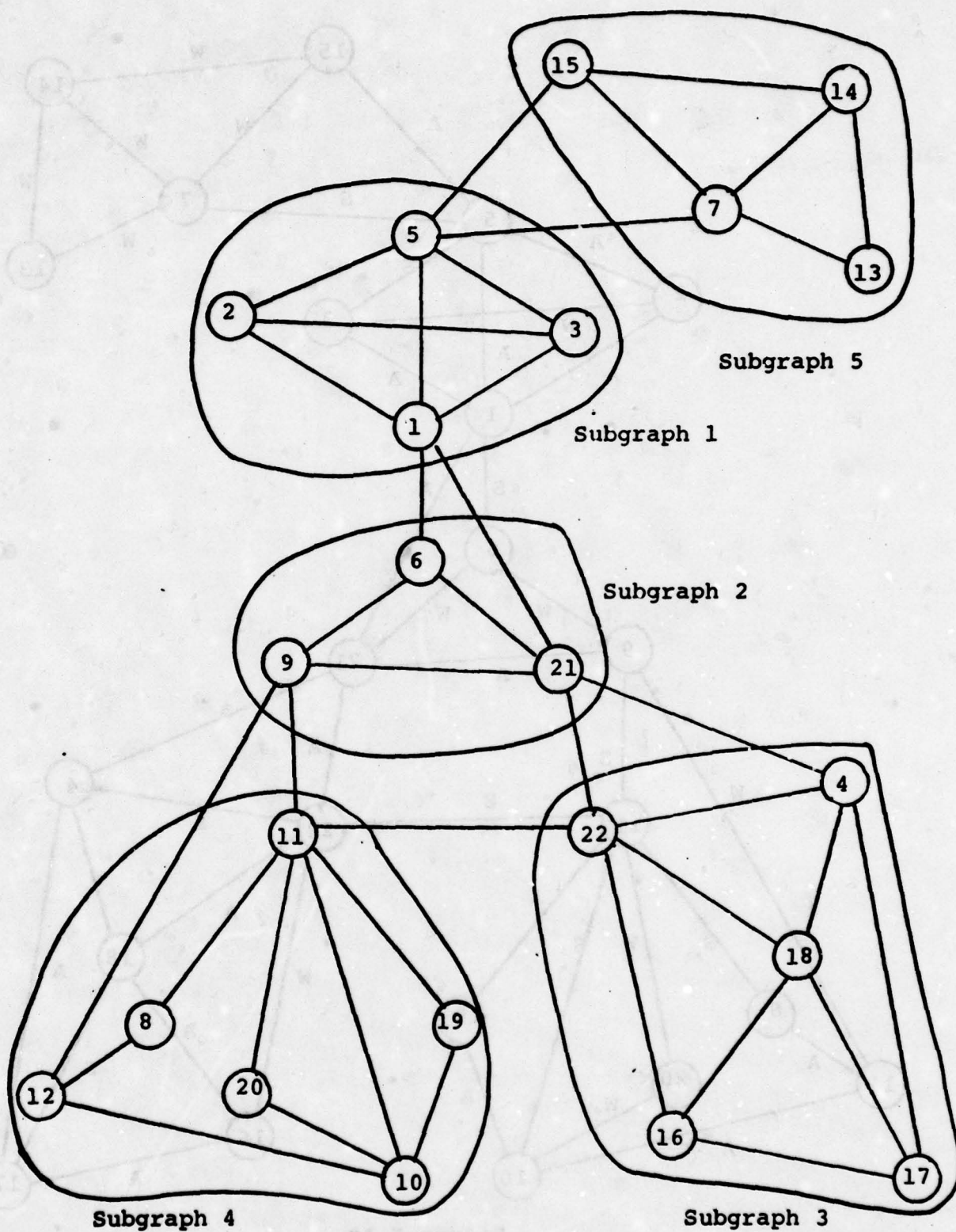


Figure 5.13

Best decomposition of the unweighted 22-node graph

Thus it is seen that the interchange method produced the best decompositions. The clusters resulting from its execution were:

<u>CLUSTER</u>	<u>NODES TO BE INCLUDED</u>
1	1,2,3,5,6
2	4,16,17,18,21,22
3	8,9,10,11,12,19,20
4	7,13,14,15

This clustering is illustrated in Figure 5.14.

The difference between the two results resides entirely in the treatment of nodes 6, 9, and 21. In the earlier case, these three nodes were lumped together to form their own cluster. However, in the present analysis, each node has been moved into one of the other clusters. Comparison between Figures 5.13 and 5.14 indicates the reason for the change: the difference resides in the relative link weights on the links extending from nodes 6, 9, and 21. It may be seen that, by partitioning in the manner illustrated in Figure 5.14, two links of "W" weight and two of "A" weight are cut. If the partition of Figure 5.13 were imposed, however, two "S" links, three "A" links, and one "W" link would have been cut. Clearly, latter partitioning is more "costly" in terms of link cuts. If the strength of subgraph (6,9,21)

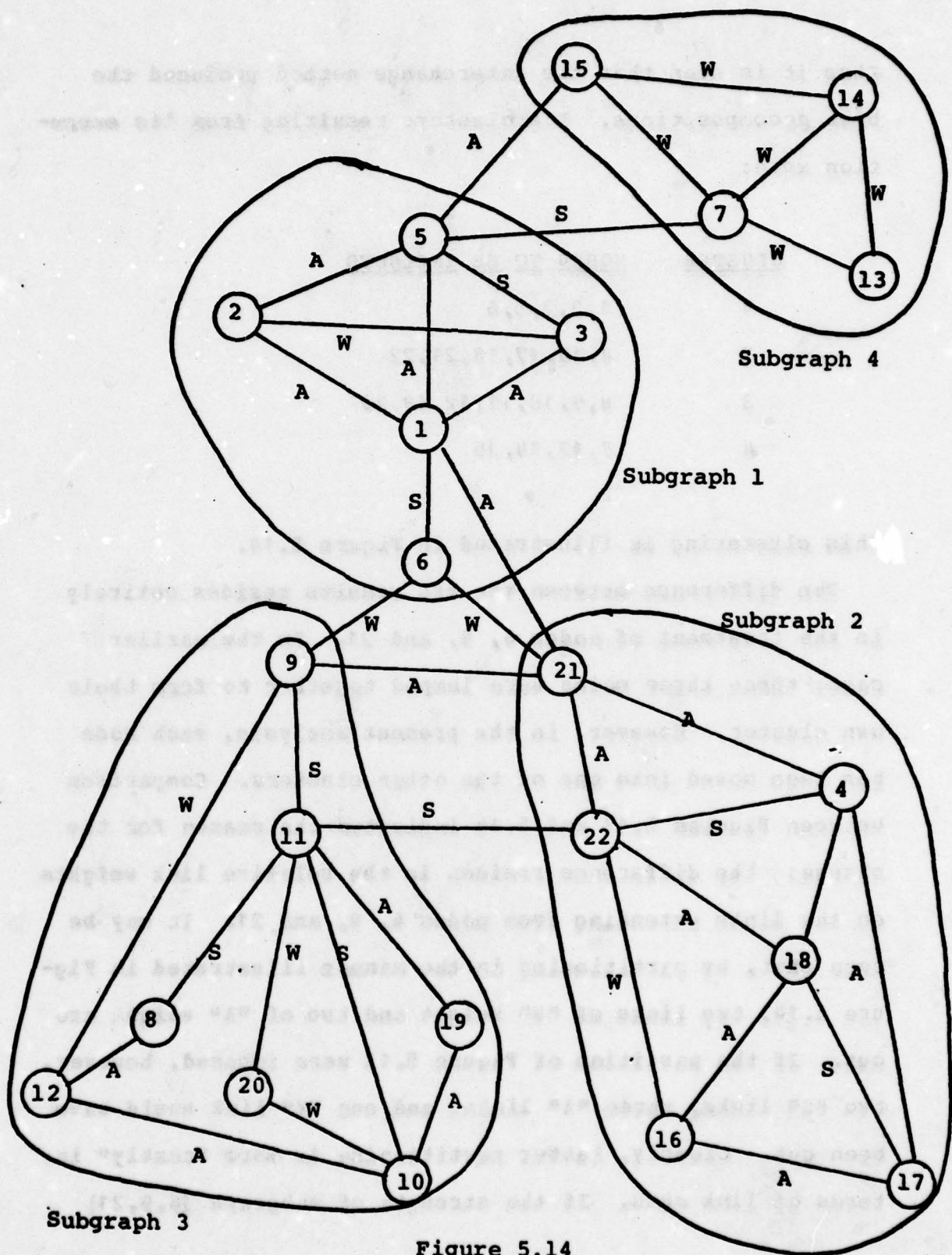


Figure 5.14

Best decomposition of the weighted 22-node graph

was quite high, this more costly set of link cuts might have "paid off"; however, as the links (6,9), (6,21), and (9,21) are not particularly strong (having weights of W, W, and A respectively), such is not the case. Consequently, the new decomposition produced by the extended analysis techniques can in this case be reasonably justified as being an improvement over the earlier approach.

As part of the documentation of the SDM analysis package included in the appendices, the terminal session that produced the foregoing results for the 22-node graph is included as Appendix E.

5.5 SDM ANALYSIS USING OTHER MODEL EXTENSIONS.

To this point we have addressed in some detail the question of how to extend the key analytical mechanisms used in performing SDM decomposition analysis to incorporate interdependency strength factors. However, interdependency strength is not the only extension to the SDM representational model proposed in Chapter 4. Other proposed extensions include

1. interdependency similarity relationships and accompanying strength factors;
2. implication relationships between requirements and between interdependencies;
3. hierarchical implication relationships.

That chapter made it clear that the various kinds of model extensions that were proposed there were not necessarily appropriate bases for extensions to the full set of decomposition techniques. To take a case in point, it may be clear that directed (implication) relationships between requirements exist, are relatively easily identified by systems analysts, and may be appropriately represented in the SDM model. It may not, however, be at all obvious how the information contained in such relationships ought to be used to affect good decompositions of the requirements graph.

Study of the kinds of model extensions identified in Chapter 4 has suggested that some of the proposed extensions may be more generally relevant than others. In particular, application of all the proposed extensions to a particular case study, the 22-requirement DBMS discussed in the previous section, indicated that, of all of the proposed secondary extensions, the one most frequently and usefully applied was interdependency similarity relationships. The DBMS example discussed there gave rise to ten such relationships, whereas only three inter-requirement implication relationships could be identified, and only one hierarchical implication relationship identified.

On the basis of this example, and of the broader insight gained in studying it and other similar sets of requirements (especially the application study discussed in Chapter 7), it may be tentatively concluded that, if the SDM analysis techniques are to incorporate any of these "secondary" extensions to the main weighted-graph techniques reported earlier in this paper, then attention should be directed toward interdependency similarity relationships first. The purpose of this section is to explore possible ways in which this may be accomplished.

5.5.1 Interdependency Similarity Relationships.

To briefly review the nature of interdependency similarity relationships (hereafter termed "ISR's"), the following is reproduced from Chapter 4:

"Two or more links (interdependencies) may represent the same, or closely related implementation issues. A simple example of this possibility is illustrated in Figure 5.15. The links joining requirements 1 and 2, and 2 and 3, both represent the interdependency "ISAM organization," an implementation consideration through which both requirement pairs (1,2) and (2,3) are deemed by the designer to be interdependent.

In this example, the two links represent entirely the same implementation issue. In general, the degree of "sameness" between two or more implementation issues will generally be less than 100 percent in the eyes of the designer, due to the inherent fuzziness in the specification of both functional requirements and implementation schemes. The judgment as to whether a given pair of links "really" represent the same implementation issue is, again, a designer decision.

Going one step further, a weight factor could be associated with the similarity assessment to represent the extent to which the designer judges the two implementation issues to be the same. That is, such a weight would correspond to the extent of overlap between the implementation issues, in the designer's estimation."

The question at this point is not what ISR's are, not how they ought to be logically represented or viewed, but rather how they may be incorporated into the decomposition analysis. Two different approaches present themselves, both of which have been investigated.

5.5.1.1 Modification of Similarity Coefficients.

The main effect of ISR's is related to the effect of interdependencies themselves (essentially, an ISR may be viewed as an "interdependency between two interdependencies"; alternately, they may be likened to Chen's concept of "relationship relations" (see Chen 77)). Whereas the importance of interdependencies is to suggest that the associated requirements be grouped together in a decomposition, the proper interpretation of an ISR is to suggest that the associated interdependencies are related and ought to be grouped together. However, since interdependencies are not "things," this statement has to be taken to mean that the system requirements which correspond to the related interdependencies ought to be grouped together.

The foregoing is actually harder to say than illustrate. Figure 5.16 indicates that interdependencies (1,2) and (2,3) are related. A good decomposition algorithm ought, therefore, to operate so as to group (1,2) and (2,3) into a common subgraph - i.e., to group requirements 1, 2, and 3 together. This is not to say that such a grouping must occur, of course, only that our preference for such a decomposition would be stronger than would be the case were there ISR no so assessed.

One possible approach to adjusting the decomposition algorithms to take account of this issue would be to modify

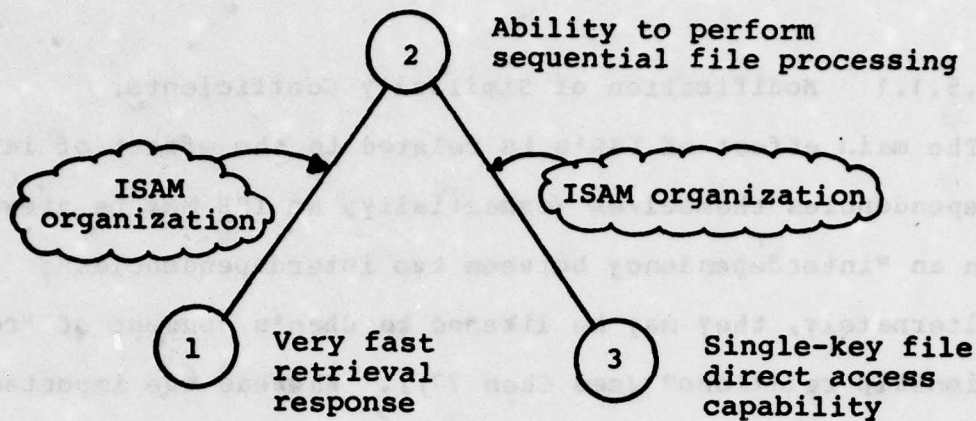


Figure 5.15

A simple interdependency similarity relationship (ISR)

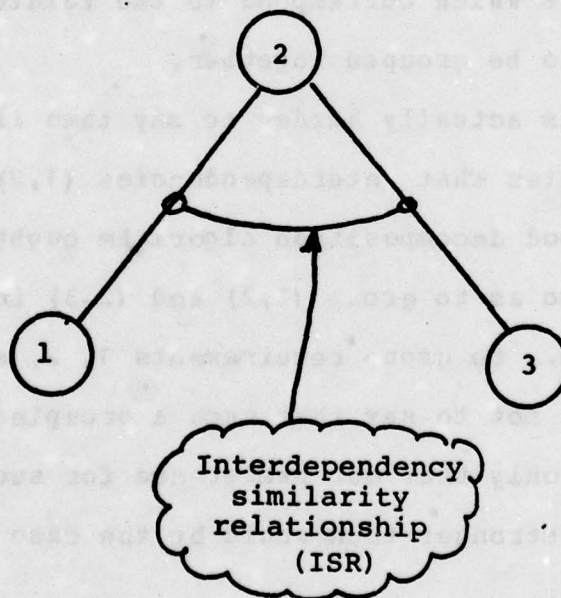


Figure 5.16

Representation of the ISR of Figure 5.15

the particular similarity coefficients associated with the affected requirements. In the foregoing example, the similarity coefficients ($p_{12}, p_{13}, p_{23}, p_{21}, p_{31}, p_{32}$) would all need to be modified - i.e., increased - so as to reflect our judgment of the extent to which the ISR makes these three requirements "more similar" to each other than they otherwise would be.

As usual, there is no objective rule to be followed, and we must be guided again by our intuition. However, some considerations to be kept in mind include:

1. however much the increase in the particular coefficients, they should not be increased beyond 1.0, the accepted upper limit for p_{ij} ;
2. a weight factor may be attached to the ISR, in a manner parallel to that for interdependency strengths, and may be used in the adjustment of the similarity coefficients.

A reasonable similarity modification technique, which observes the two conditions given above, may be stated as follows. Define the modified similarity between the affected requirements to be

$$p'_{ij} = \min \{ 1.0, p_{ij} * (1.0 + v) \},$$

where i and j range over the three (or possibly four) affected node pairs, and v is the weight on the associated ISR.

For instance, if v were taken to be 0.5, the suggested modification would increase each affected similarity coefficient by 50 percent, to a maximum of 1.0.

This similarity modification approach has been incorporated, for testing purposes, into the SDM analysis package. Its effectiveness as compared to a different approach (to be discussed momentarily) will be reported below.

One serious shortcoming of the similarity modification approach is that its effect is brought to bear only through the use of the hierarchical clustering algorithms. It is not driven by a modification to the underlying graph itself. Another shortcoming is that its impact is not reflected in the decomposition goodness measure M , since M depends only on the underlying graph structure and the particular decomposition at hand, not on the inter-node similarities. The only real impact of this approach is to guide the clustering process along a (possibly) different, presumably better, path than would otherwise be the case. We will see that the second suggested approach, explored next, manages to avoid these drawbacks.

5.5.1.2 Modification of the Graph Structure.

The major drawbacks to the first approach to incorporating ISR information into the decomposition process hinged on the fact that only the similarity coefficients, not the underlying graph structure, was impacted.

If we study the underlying structure, it is clear that what is needed is a modification that will transform each ISR into a mechanism that serves to more strongly "hook together" the corresponding requirements nodes that would otherwise occur. A simple solution is to transform each ISR into a new graph node, with links to each connected requirement node. Since these new nodes do not represent original requirements, but rather ISR's, they are termed "ISR-nodes." Examples of ISR-nodes are given in Figure 5.17(a) and (b).

The ISR-node approach does meet all the important requirements for incorporating ISR information into the analysis:

1. since ISR-coupled nodes are now more strongly bound together, through the medium of the new ISR-nodes, decompositions will be more likely to group such nodes together than otherwise;
2. since this technique modifies the underlying graph structure, all decomposition methods, including the interchange technique, continue to apply;
3. the goodness measure will reflect the impact of ISR information.

The ISR-node approach seems the preferable method, and this is borne out in a case study, reported next.

5.5.2 An Example of the Use of ISR Data.

For a case study, we will consider again the set of 22 DBMS requirements studied earlier. Interdependency similar-

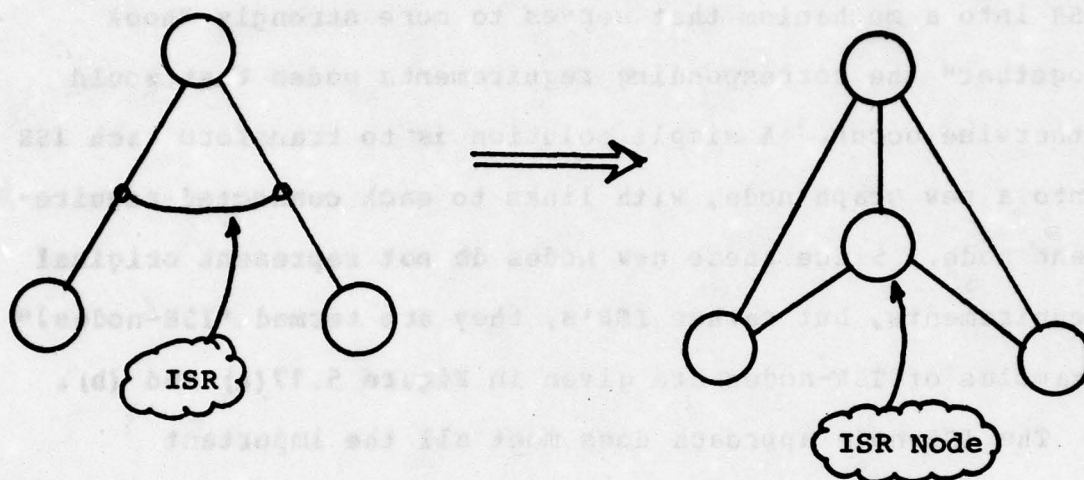


Figure 5.17(a)

Modification of a 3-node subgraph to include an ISR node.

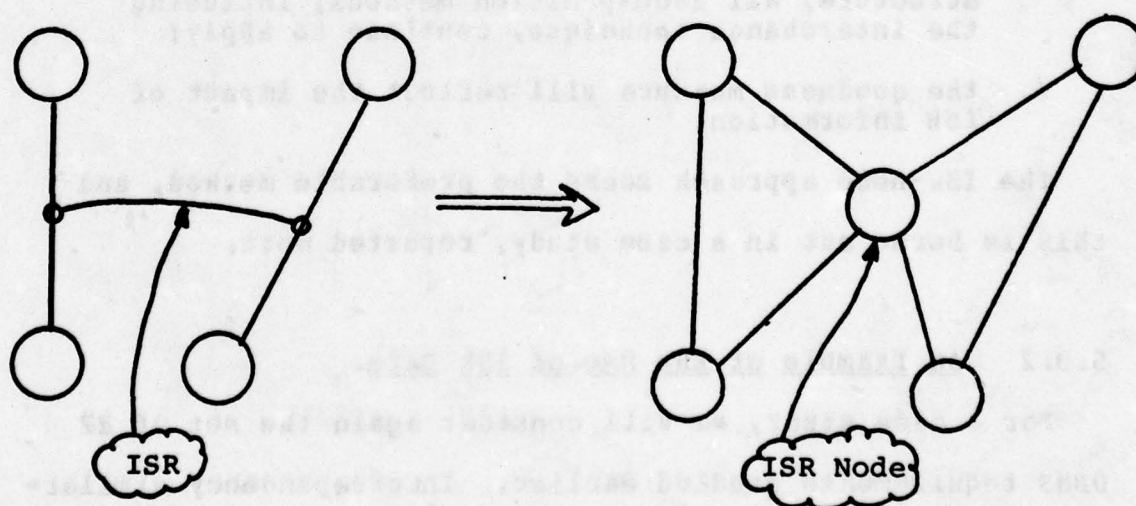


Figure 5.17(b)

Modification of a 4-node subgraph to include an ISR node.

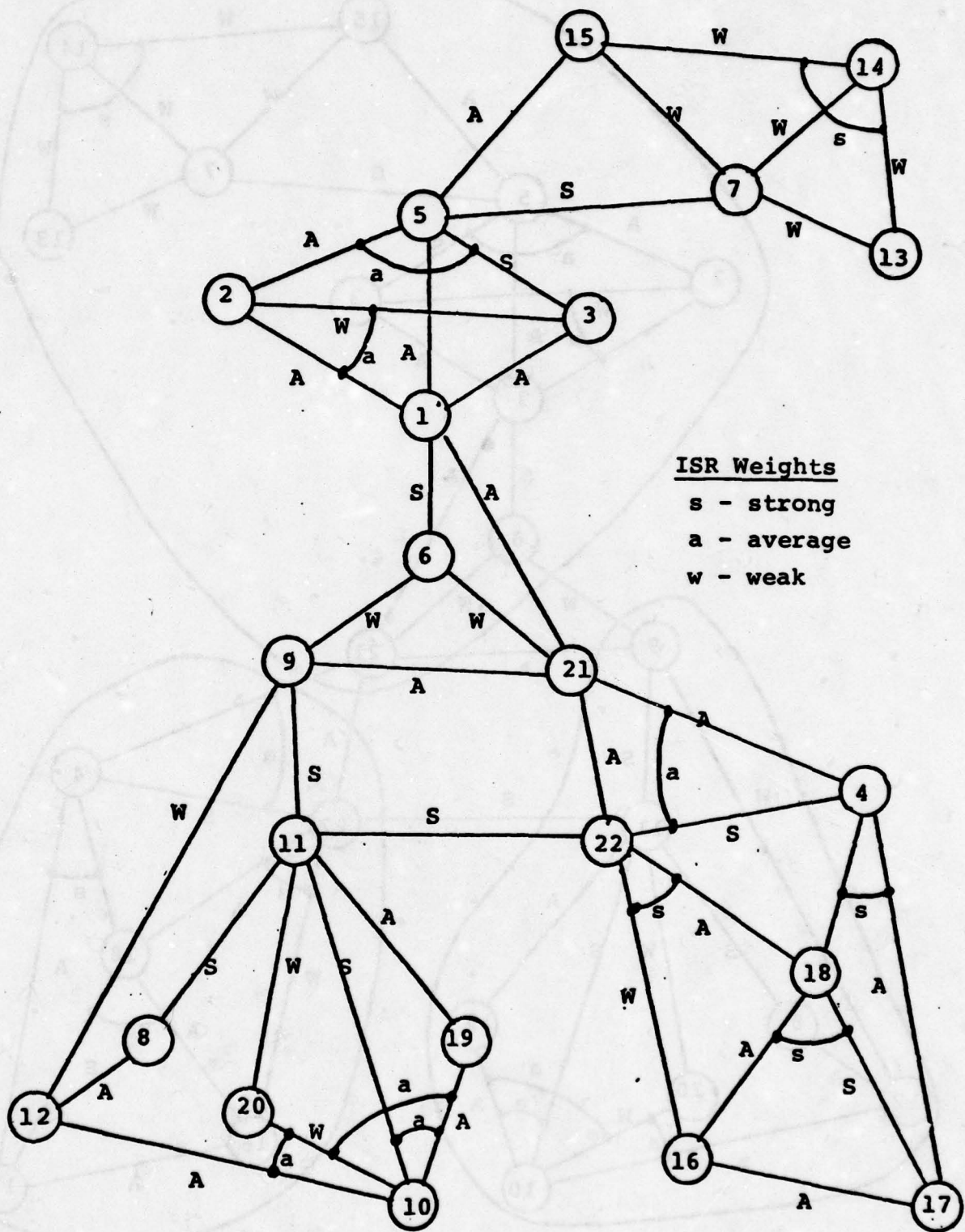


Figure 5.18
 The 22-node requirements graph including ISR's

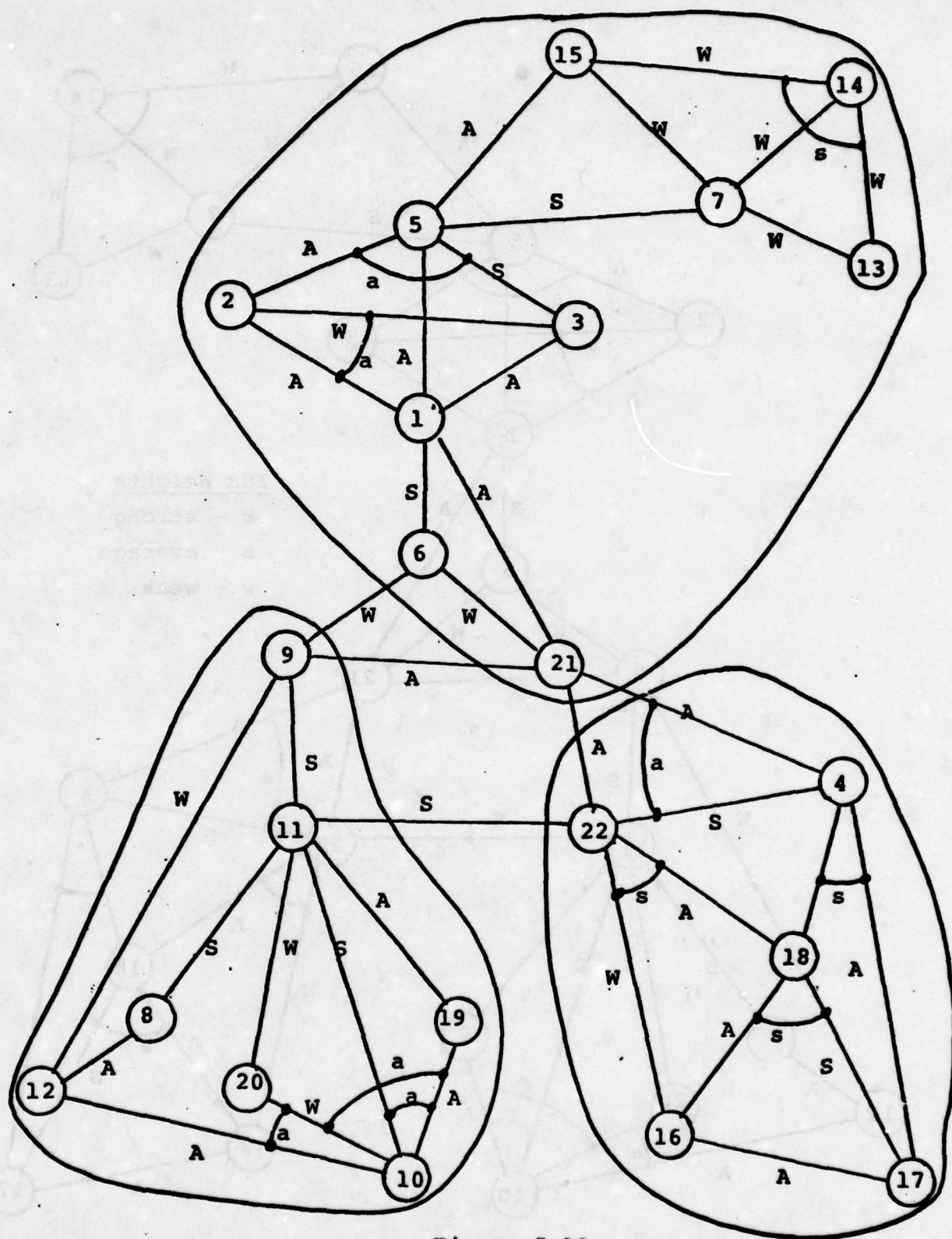


Figure 5.19

Best clustering of 22-node graph with
ISR's not included.

ity relationships (ISR's) were assessed for the requirements graph in Chapter 4, and associated strength factors were assigned. Figure 5.18 contains the relevant information in graphical form. Further details are available in Chapter 4.

In the first part of the test, the original graph structure (with no ISR-nodes) was input to the SDH analysis package, and the best decomposition located using only the clustering algorithms. This turned out to be:

<u>CLUSTER</u>	<u>REQUIREMENT NODES</u>
1	1,2,3,5,6,7,13,14,15,21
2	4,16,17,18,22
3	8,9,10,11,12,19,20

For this decomposition, $M = 0.33$. This decomposition is illustrated in Figure 5.19

Then the MODSIM command (not described in Appendix D, as it is still experimental) was executed, and the appropriate similarity coefficients modified as discussed earlier. A second decomposition analysis resulted in a somewhat different decomposition, namely:

<u>CLUSTER</u>	<u>REQUIREMENT NODES</u>
----------------	--------------------------

1	1,2,3,5
2	7,13,14,15
3	6,9,21
4	4,16,17,18,22
5	8,10,11,12,19,20

The goodness measure turned out to be $M = 0.20$. The decomposition is shown graphically in Figure 5.20.

These results are reasonable and believable, as is also suggested by Figure 5.18. The inclusion of the assessed ISR's ought to move the optimal decomposition in the direction of four or five "clumps," rather than the three obtained earlier. However, since the underlying graph was not modified, it is not surprising that M is somewhat lower in the second case.

The second test involved changing the graph structure to include 10 new ISR nodes, then analyzing the resulting 32-node graph in the usual manner. The optimal result obtained from this analysis is slightly different (and, we will argue, somewhat preferable) from that obtained in the first (similarity modification) analysis. It is:

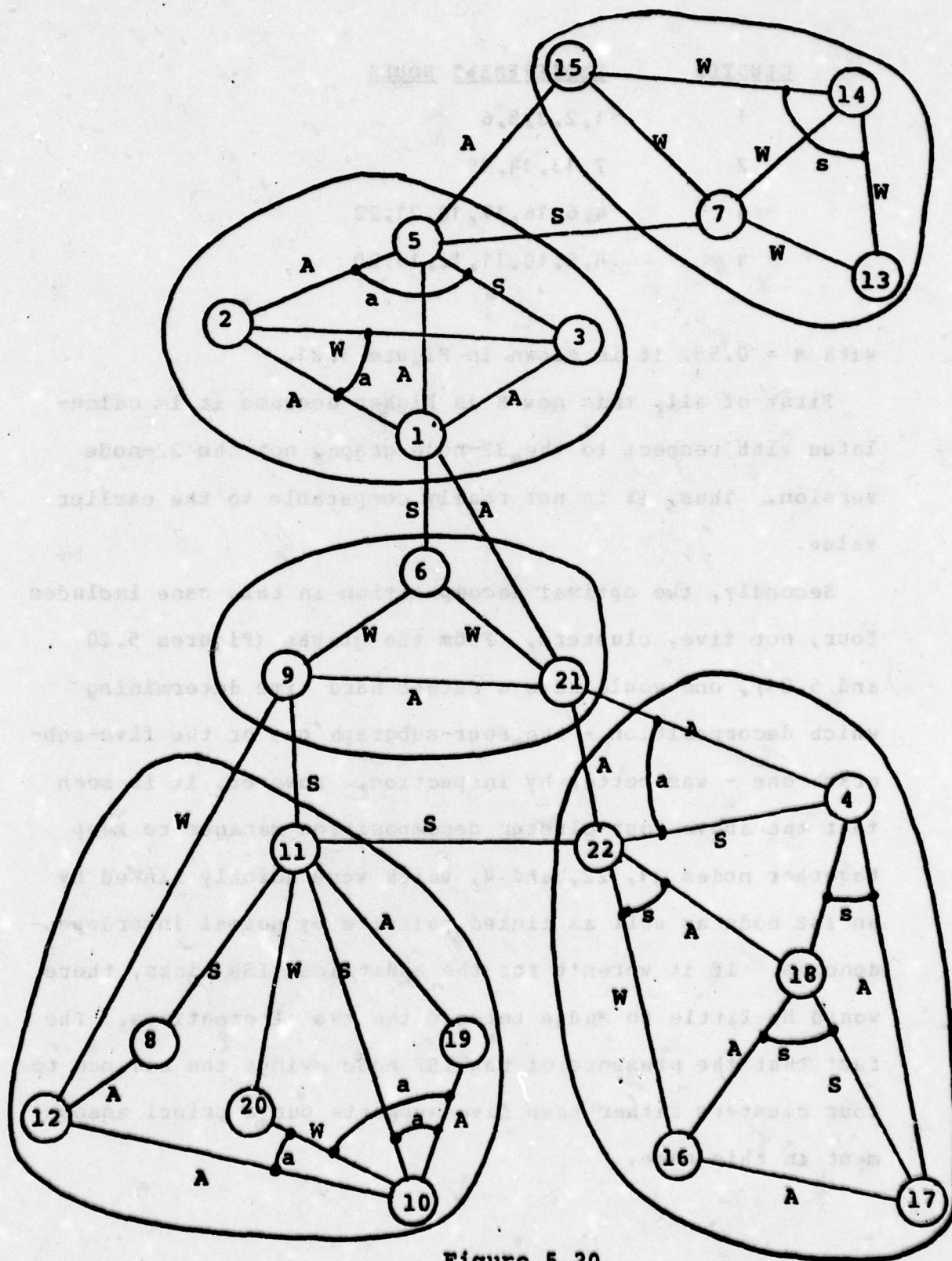
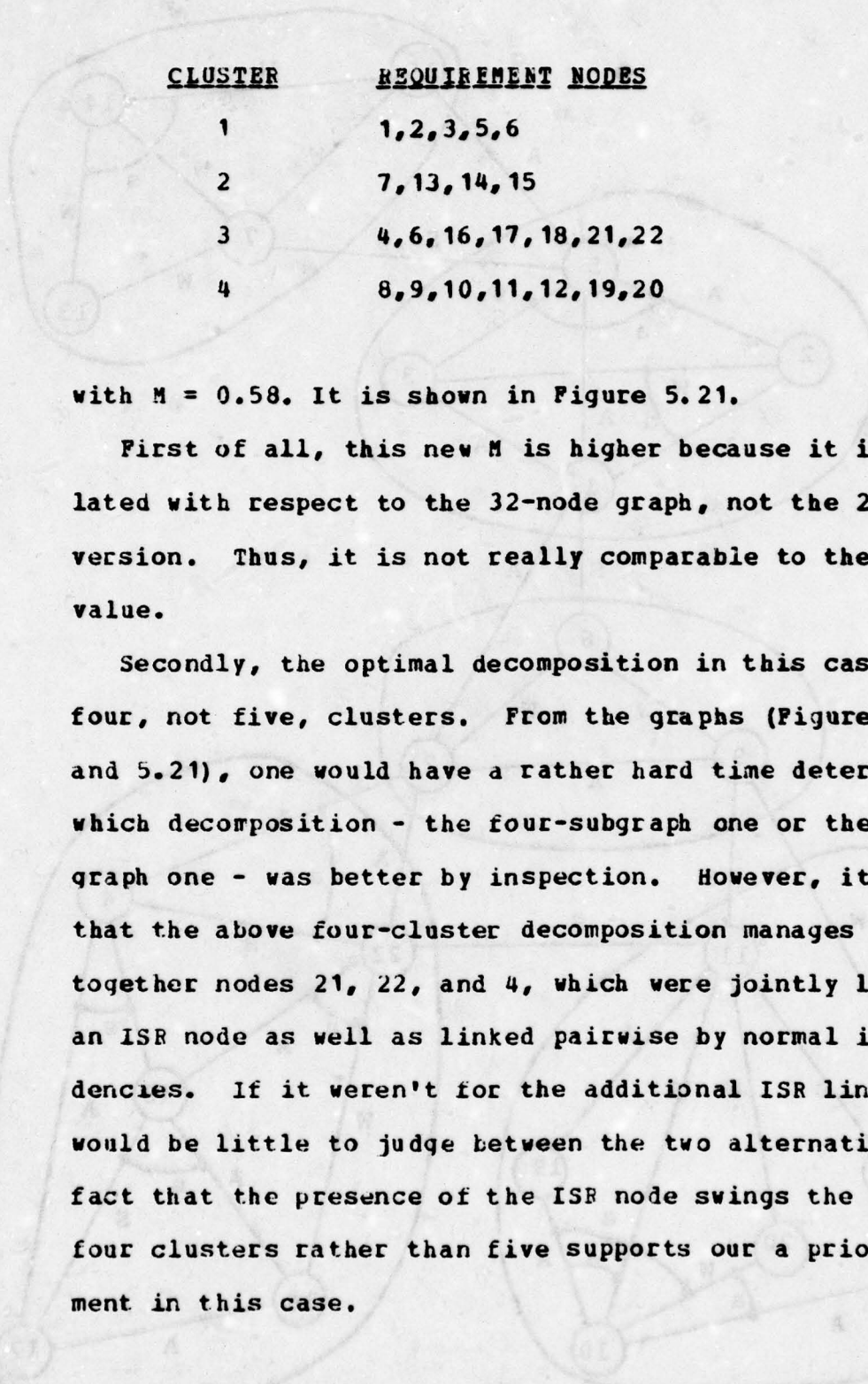


Figure 5.20

Best cluster decomposition of 22-node graph using the similarity modification approach to treating ISR's



<u>CLUSTER</u>	<u>REQUIREMENT NODES</u>
1	1,2,3,5,6
2	7,13,14,15
3	4,6,16,17,18,21,22
4	8,9,10,11,12,19,20

with $M = 0.58$. It is shown in Figure 5.21.

First of all, this new M is higher because it is calculated with respect to the 32-node graph, not the 22-node version. Thus, it is not really comparable to the earlier value.

Secondly, the optimal decomposition in this case includes four, not five, clusters. From the graphs (Figures 5.20 and 5.21), one would have a rather hard time determining which decomposition - the four-subgraph one or the five-subgraph one - was better by inspection. However, it is seen that the above four-cluster decomposition manages to keep together nodes 21, 22, and 4, which were jointly linked by an ISR node as well as linked pairwise by normal interdependencies. If it weren't for the additional ISR links, there would be little to judge between the two alternatives. The fact that the presence of the ISR node swings the balance to four clusters rather than five supports our a priori assessment in this case.

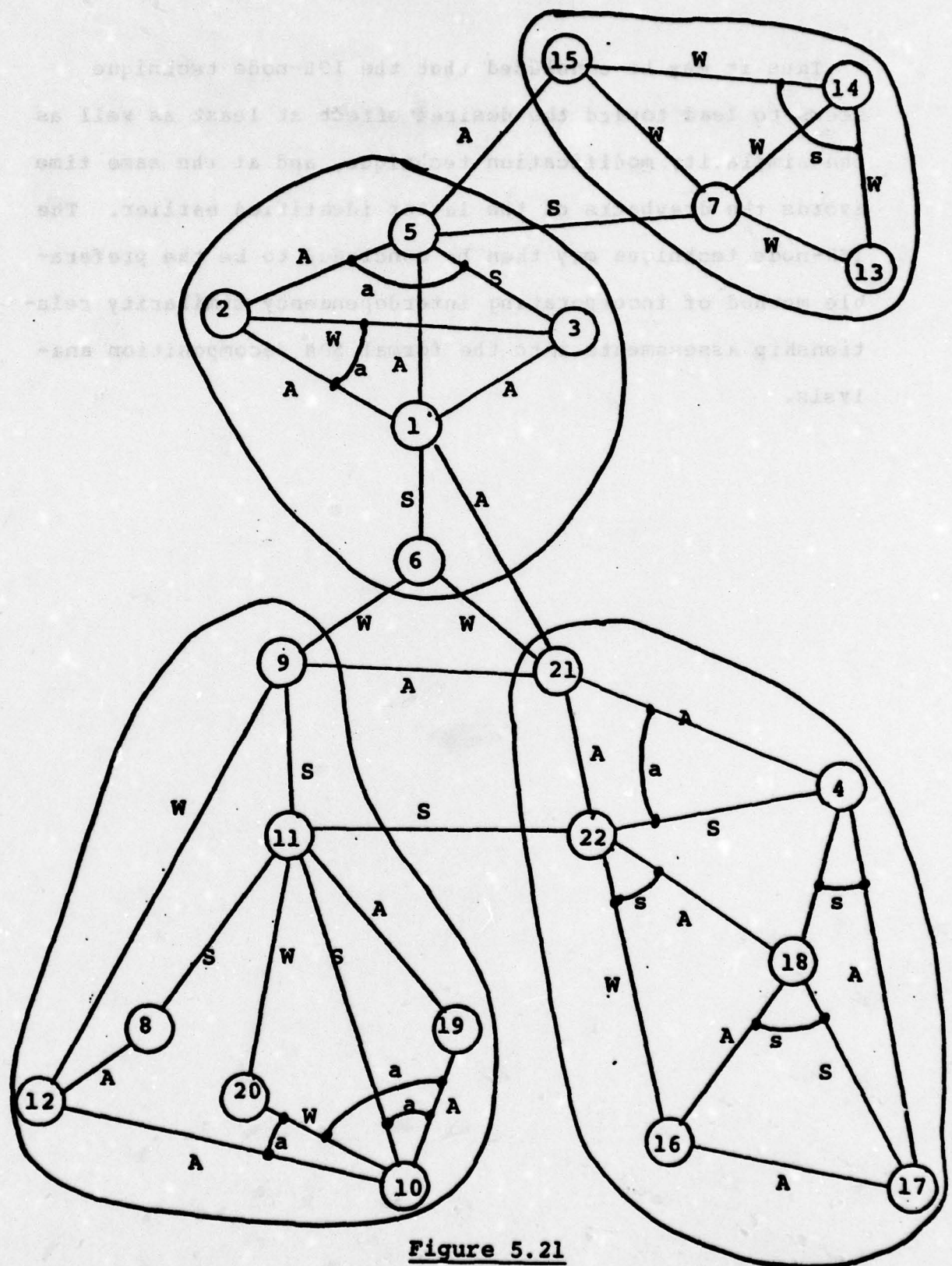


Figure 5.21

Best decomposition of the 22-node graph using the ISR-node approach to treating the ISR data.

Thus it may be concluded that the ISR-node technique seems to lead toward the desired effect at least as well as the similarity modification technique, and at the same time avoids the drawbacks of the latter identified earlier. The ISR-node technique may then be concluded to be the preferable method of incorporating interdependency similarity relationship assessments into the formal SDM decomposition analysis.

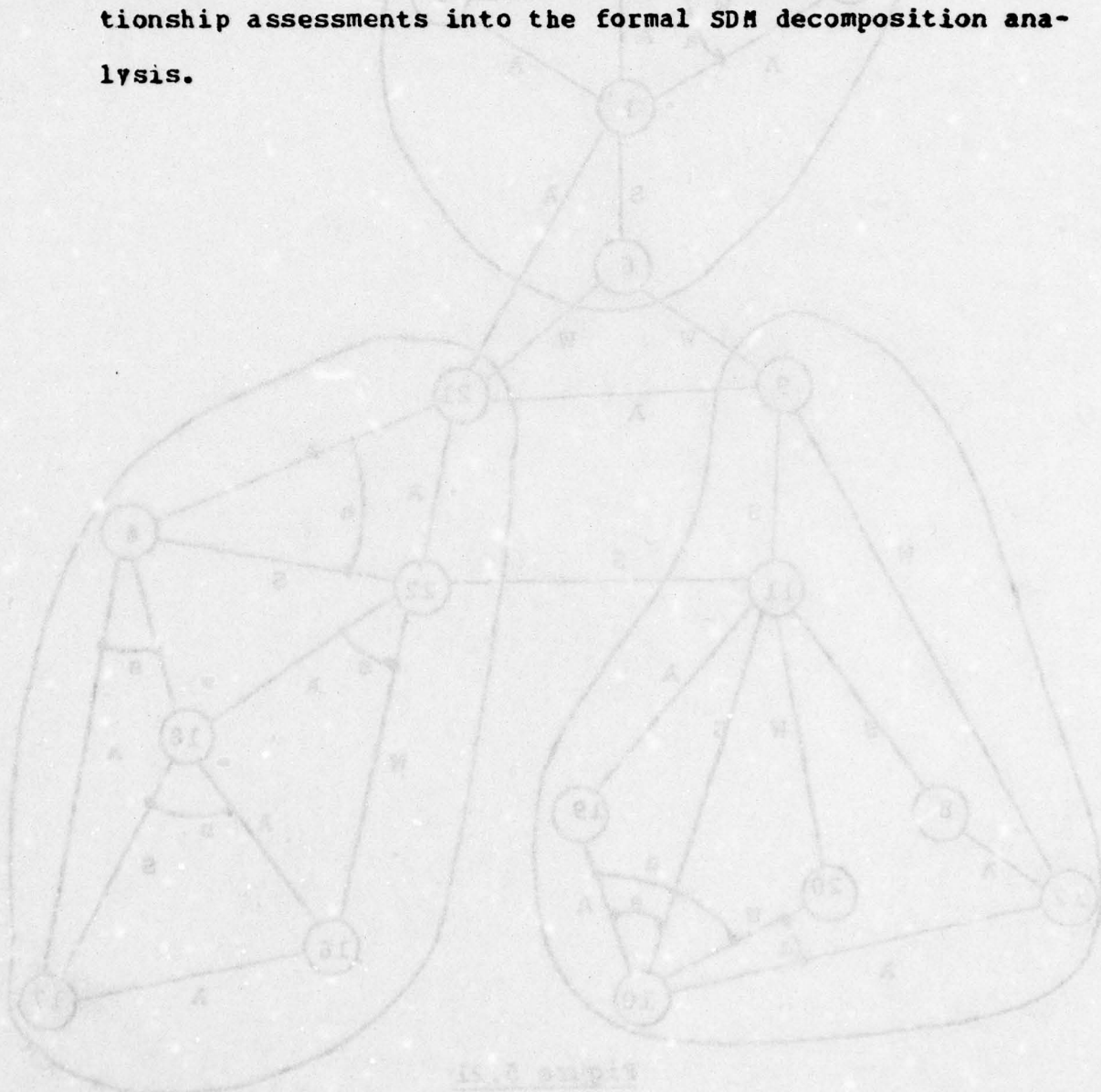


Figure 5.11

Best decomposition of 7-node graph using the ISR-node approach to testing the ISR data.

5.6 SUMMARY.

Central to the Systematic Design Methodology is the graph model used to represent the design-relevant information pertaining to a target system. A key issue in the development of this methodology is the determination of what information ought to be elicited from a system architect to use in the creation of a preliminary design. The issue is essentially one of cost effectiveness: what is the cost (primarily in designer time and effort) of attempting to elicit a particular piece of information, and what can such information add to the effectiveness of the design?

While easily posed, this question is, at this point, impossible to answer precisely. While the cost side of the issue is not too difficult to deal with, the really difficult problem resides in determining design quality, and in particular, the impact that certain information may have on design quality.

The approach followed within the SDM project has been to use the surrogate "high strength-low coupling decomposition of the system requirements graph" for the real objective, "high design quality." While there are some very believable arguments supporting the appropriateness of this surrogate ((Alexander 64), (Andreu 78)), the case is far from com-

plete. Of course, the same can be said for countless other developments wherein the cost of full-scale objective testing is prohibitively high (including essentially all other software design and development methodologies).

In this research, then, we have placed considerable faith in our own intuition and judgment for effecting a reasonable tradeoff between what design-relevant information may be elicited from designers at a reasonable "cost" and what information is most useful and effective in creating a better preliminary problem structuring. To that end, certain extensions to the basic binary-link representational model used in earlier SDM studies, were proposed and examined in Chapter 4. In this chapter, the SDM analysis mechanisms were also extended, to incorporate two of the most important model extensions: interdependency strength assessments, and interdependency similarity relationships. The manner in which these extensions impact the SDM decomposition goodness measure M , the inter-requirements similarity calculations, the clustering algorithms, and other aspects of the analysis scheme, has been described, with examples, herein. As well, the appendices include accompanying documentation of the SDM analysis package.

* * * * *

In the next chapter we consider in detail a new decomposition algorithm for partitioning weighted requirements

graphs. Under comparison with the hierarchical clustering techniques described in this chapter, it turns out to be considerably more effective for the purpose of locating a decomposition with high strength and low coupling. Following that, we present the result of the application of the Systematic Design Methodology to a real design problem. Then in Chapter 8 we address the issue referred to above - how are we to determine whether SDM has value for software system design?

Chapter VI

SDM DECOMPOSITION ANALYSIS USING THE INTERCHANGE ALGORITHM.

6.1 INTRODUCTION.

This chapter introduces a new algorithm for partitioning weighted graphs in a top-down hierarchical manner. The algorithm, termed the interchange partitioning technique, has been developed to aid in the analysis of requirements graphs generated through the Systematic Design Methodology. However, it is sufficiently general and powerful to be of use in numerous other types of graph analysis problems also.

As well as describing and giving examples of the basic interchange technique, this chapter also included a discussion of certain simplifications that may be made to the algorithm in order to significantly improve its operational efficiency without hampering its effectiveness (ability to produce good graph decompositions). The algorithm is compared in effectiveness against the hierarchical clustering methods discussed earlier (Chapter 5), and is shown to be significantly more effective in locating good graph partitions. Also, a "master control" algorithm is presented for guiding the execution of a complete graph decomposition using the interchange partitioning technique. Appendix F

contains a computer-produced execution trace of the algorithm operating on an example graph. Examples of the use of the interchange partitioning scheme are included throughout.

6.1.1 Graph Decomposition in the SDM Context.

There are two distinct issues involved in the graph decomposition problem. First, there is the question of what the decomposition objective function ought to be. Second, there is the question of how to go about actually effecting the decomposition - i.e., how to identify and select subgraphs.

The notion of a "good" graph decomposition is, of course, context dependent. In the development of the SDM, we have adopted a measure that operationalizes a key concept of software design, namely, that a good software architecture is one which both maximizes the internal strength of each system module, and which also minimizes the coupling between modules. Thus a graph decomposition in which each subgraph is densely connected, and pairs of subgraphs are loosely interconnected, will have a relatively high value of M , the objective function. This objective function is discussed in greater detail in Section 6.4.1.

As for actually effecting the decomposition of graphs, we have experimented with a variety of techniques. These techniques may be classified into two major categories:

1. clustering techniques (bottom-up), and
2. partitioning techniques (top-down).

Clustering techniques require the generation of a similarity (or dissimilarity) matrix to express the closeness (or distance) between all pairs of data points. In the case of graph decomposition, the "data points" are the nodes of the graph. Once a (dis)similarity matrix has been defined, various hierarchial clustering heuristics may be applied to successively lump together individual nodes into subsets, subsets into larger subsets, etc., until the entire set is generated. While hierarchical clustering techniques as such have no inherent stopping rule, the goodness measure, M , may be calculated after each subset merger, and the particular decomposition exhibiting the highest M "remembered" as the best decomposition. Since the focus of this chapter is partitioning algorithms, not clustering techniques, the reader is referred to Chapter 5 and the references for further details.

Partitioning techniques take a variety of forms, but have the common property of dealing directly with the graph structure, rather than a similarity matrix defined out of the graph structure. Typically, partitioning techniques seek to successfully break up a graph into subgraphs until either some stopping criterion is reached, or until each subgraph contains a single node.

Partitioning techniques are especially useful in SDM decomposition for two different reasons. First, since they are fundamentally different from the various types of clustering techniques also used for this function, they provide an effective cross-check on the validity of the results achieved by the other methods. Also, they tend to locate a best decomposition with a smaller number of steps than clustering techniques, because optimal SDM decompositions occur fairly near the top of the decomposition "tree" (i.e., fairly near to the single subgraph state - see Figure 6.15).

While the graph theory literature reports a variety of approaches to the partitioning problem, none of these schemes are directly appropriate for our class of problems. For example, one common partitioning technique involves identifying complete subgraphs, then using these subgraphs as "leader" subsets, and effecting a final partition by assigning the remaining nodes to one of the leader subgraphs. However, in the SDM context, design problem representations yield graph structures with few if any complete subgraphs of non-trivial size. Also, when complete subgraphs do exist, they often overlap, and there is no obvious way of dealing with this problem in the SDM context.

6.1.2 Shortcomings of Previous SDM Partitioning Techniques.

In his work on the development of the phase-1 SDM, Andreu (Andreu 79) identified and tested two different partitioning techniques - one a "leader" identification technique similar to the complete subgraph method mentioned above, the other an iterative technique for factoring out subgraphs. These two partitioning techniques exhibit certain difficulties and shortcomings, both inherently and with respect to the present SDM model.

The leader technique, while quite efficient, serves only to identify certain "good" starting subgraphs. The number and size of the subgraphs can be controlled only grossly. Also, the technique generally results in numerous unassigned nodes, and there is no obvious means of deciding what to do with the leftover nodes: whether they ought to be assigned to certain of the leader subgraphs, and if so, which one, or whether certain of the unassigned nodes ought to be grouped together to form another subgraph.

Andreu chose to assign each of the leftover nodes to a subgraph such that the measure M increased the most. This resolution technique is not as straightforward as it sounds, however. For example, suppose the leader technique gives rise to five leader subgraphs plus ten unassigned nodes. In order to determine the leader subgraph to which the first

unassigned node (say, node x) should be assigned, the values $\Delta M_{ix} = M' - M_{ix}$ must be calculated, where M' is the goodness measure of the initial partition, and M_{ix} is the goodness measure for unassigned node x placed in leader subgraph i . Node x would then be assigned to that leader subgraph corresponding to the maximum ΔM_{ix} over all i . However, the values M' and M_{ix} must be calculated with a number of nodes unassigned. A problem arises as to how to treat these unassigned nodes: they could be treated as individual subgraphs, could be ignored, or possibly handled in yet another fashion. Each such treatment has drawbacks: as the final node resolution problem is akin to solving a set of simultaneous equations, such stepwise approaches are not directly suitable.

In particular, a simple hill-climbing resolution approach for assigning leftover nodes, such as that used by Andreu, is especially questionable, being so sensitive to the order in which the unassigned nodes are dealt with, among other things. The ineffectiveness of a hill-climbing approach in decomposition by clustering is discussed further in Chapter 5, Section 5.3.3.. The foregoing points illustrate some of the difficulties associated with leader subgraph approaches in general.

Andreu's iterative approach is novel and quite interesting, but suffers from a drawback even more severe: its

computational requirements grow very rapidly with the size of the subgraph. The procedure effectively involves iterative recomputation of a matrix of size n (where n = the number of graph nodes). Each recomputation requires n calculations. After a number of iterations, the matrix tends to stabilize in such a way that one or more subgraphs with high internal strength may be identified and "factored out" of the graph. The process is then repeated, as many times as necessary, successively identifying and factoring out subgraphs, until only a single unfactorable subgraph remains. Andreu tested the iterative approach on a few fairly small graphs, with promising results. But he also indicated that the calculation barrier rapidly becomes large as n is increased, since both the amount of calculation at each iteration (proportional to n^2) increases rapidly, and the number of iterations required to completely factor the graph also increases rapidly (by an unspecified amount). Andreu himself found the iterative approach to be too inefficient to be used for problems of non-trivial size.

Yet another drawback to both leader subgraph and iterative approaches is that they give the user very little in the way of control over the size of the resulting subgraphs. It would be useful, for example, to be able to specify, a priori, certain minimum or maximum sizes on subgraphs. For instance, a system designer might prefer a decomposition

with approximately balanced subgraph sizes and slightly lower objective function value. Providing him with control on subgraph size would allow him to make such a tradeoff intelligently. The leader technique could conceivably be modified to include such control, although there is no apparent way to so modify the iterative technique.

Finally, it is important to note that both the leader and iterative techniques were developed to partition binary (unweighted) graphs. Since the basic graph model has now been extended to include, among other things, weights on the graph links, neither algorithm is immediately applicable to the new model. Modifications for these techniques to extend their applicability to the current SDM model may be possible to develop, although such extensions have not yet been studied.

* * * * *

In the remainder of this chapter we present and illustrate a new partitioning algorithm. This algorithm, termed the "interchange" algorithm (as it functions by interchanging pairs of nodes between subgraphs), avoids the problems discussed above. It is quite efficient (we show it is polynomial time bounded); it generally reaches an optimal decomposition after a relatively small number of iterations; it produces complete partitions (thus avoids the resolution problem of the leader subgraph method); it is designed to be

applied to weighted graphs (hence can be used with the extended SDM model); and it included bounds-setting control on subgraph size. The algorithm has been implemented in PL/1, and tested on a variety of graph decomposition cases. Results of some of these tests are also reported.

6.2 THE BASIC INTERCHANGE ALGORITHM.

The graph partitioning algorithm presented here has the following features:

1. It is a hierarchical partitioning approach. That is, it seeks to partition the given graph into two subgraphs in an appropriate fashion, then to partition one of the subgraphs, etc. This approach is the inverse of that followed by the hierarchical clustering techniques mentioned earlier, in that the clustering techniques progressively group subgraphs (subsets) together, forming fewer and larger clusters, whereas the hierarchical partitioning technique successively subdivides the subgraphs, forming more and smaller clusters.
2. The basic mechanism underlying the technique is to begin with an arbitrary initial factoring of the current subgraph (the subgraph to be partitioned) into two equal-sized, smaller subgraph then to perform pairwise interchanges of nodes between the two halves according to a simple criterion until no more improvement can be made in a certain objective function. Then resulting two partitions then replace the original subgraph in the global factoring of the given graph.
3. The algorithm is quite efficient. To factor a graph of $2n$ nodes into two subgraphs of n nodes each requires a number of operations proportional to $(n^2 + n)$. Thus it is of the class of n -squared procedures, the most efficient class of general graph-manipulation procedures. (Some graph procedures belong to even more efficient classes, e.g., order n , but they are generally oriented to very specific problems or are significantly constrained in terms of their generality.)
4. The technique allows the user to specify upper and lower bounds on the sizes of the subgraphs to result from the partitioning of a given graph. In the software design context, this feature may be quite useful - for example, in cases where the designer has a priori information about the appropriate size range for a given design subproblem.

In this section we describe the basic interchange algorithm, which forms the heart of the interchange partitioning scheme. In the following sections, we describe both extensions to, and simplifications of the algorithm, a control structure for implementing the algorithm for application in the SDM, and some examples of its use.

6.2.1 The Basic Interchange Technique.

The basic interchange technique used in this algorithm is partly based on early work by Kernighan and others (Kernighan & Lin 70). It functions as follows. Given an arbitrary initial partitioning of a weighted graph with an even number of nodes into two equal sized subgraph. Pairs of nodes, one node taken from each of the two subgraphs, are repeatedly interchanged so as to improve the partitioning with respect to a given quality criterion (to be discussed shortly), until a certain stopping condition is reached. A test is required to determine whether or not there is anything to be gained by repeating the procedure another time (Kernighan & Lin 70). Extensions of the procedure to the case of arbitrary-sized initial graphs and non-equal sized subgraphs will be discussed shortly.

Central to the interchange technique is the issue of what the quality criterion (the criterion used to determine which pairs of nodes, out of all possible pairs, should be

interchanged) ought to be. In Kernighan's development, interest focused on minimum-cut partitions. That is, Kernighan sought a partitioning such that the sum of the weights on the severed links was a minimum. Actually, quite efficient heuristic procedures for the minimum-cut problem have been available for some time, one of the earliest having been presented by Ford and Fulkerson in the context of their proof of the famous "max-flow min-cut" theorem (Ford & Fulkerson 60). However, these approaches gave no control over subgraph size, one of Kernighan's (and our) objectives.

In the present case, the quality criterion clearly should be related to our concept of the global goodness of a graph decomposition - i.e., to the goodness measure, M . Accordingly, we adopt, as a measure of the "gain" realized through interchanging two nodes in a given bi-partition, the impact that the interchange would have on M . This concept of interchange gain will be employed to guide the development of the basic interchange algorithm.

The discussion to follow is best motivated by means of a simple example. Suppose we wish to determine a partitioning of the six-node graph given in Figure 6.1, into two partitions of three nodes each. Further, suppose we begin with some arbitrary initial partition, say, the one illustrated in Figure 6.2.

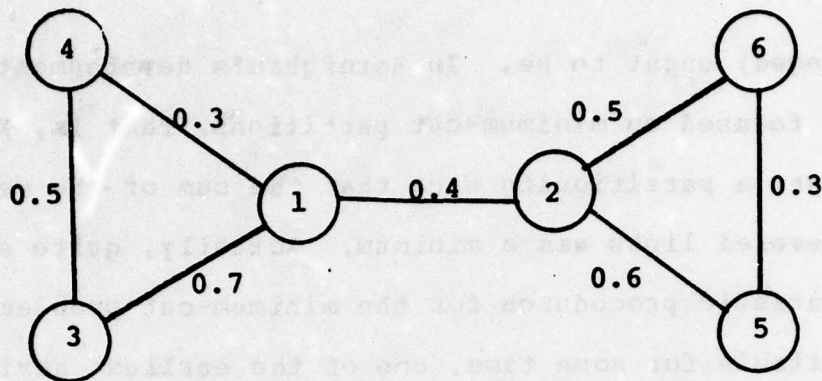
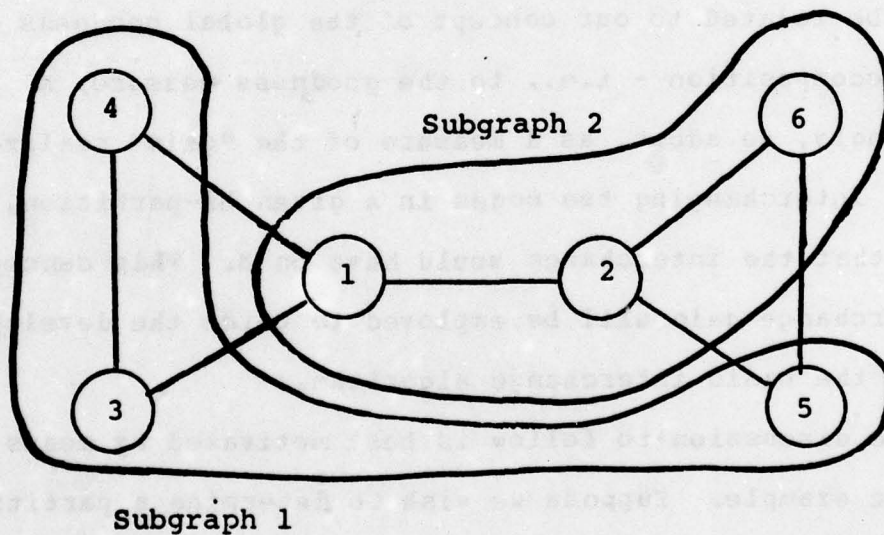


Figure 6.1

A simple weighted graph



(to minimize diagram complexity, weights on graph links will only be shown when a graph is first introduced)

Figure 6.2

(Arbitrary) initial partitioning of graph of Figure 6.1

Inspection of Figure 6.2 indicates that the interchange (① \rightarrow 1; ⑥ \rightarrow 2) (i.e., node 1 switched into subgraph 1, node 6 into subgraph 2) would produce an optimal partitioning of this graph. The interchange procedure functions as follows. First, the quantity $\Delta M(x,y)$ is calculated for each node pair (x,y) , where x belongs to one of the two subgraphs and y to the other. $\Delta M(x,y)$ is the net impact on the goodness measure M of the entire decomposition that would occur as a result of interchanging nodes x and y . Next, the specific node pair (x^*,y^*) for which $\Delta M(x,y)$ is a maximum is located, and the values x^* , y^* , and $\Delta M(x^*,y^*)$ are recorded. Nodes x^* and y^* are "marked" to be no longer eligible for interchange considerations (thereby reducing by $(2n-1)$ the number of node pairs that must be considered for interchange at the next iteration - assuming the original graph contained $2n$ nodes). This procedure is repeated again for the reduced graph, and repetition continues until no more unmarked node pairs remain. Thus if the original graph contains $2n$ nodes, arbitrarily partitioned into two subgraphs of n nodes each, there will be n repetitions of the above procedure. The end result will be a list of n triples, the k th triple being of the form

$$(x^*, y^*, \Delta M(x^*, y^*)),$$

where x^* and y^* are the nodes to be interchanged at step k , and $\Delta M(x^*,y^*)$ is the gain in global partition goodness that

would occur as a result of making the (x^*, y^*) interchange. It should be noted that the gain $\Delta M(x^*, y^*)$ at step k is really a marginal entity, as it is calculated under the assumption that the previous $k-1$ interchanges in the list were in fact implemented. Also, any partition gain value could be positive or negative, although the sum of all n values must always equal zero, since summing all n values corresponds to the total gain achieved from interchanging all node pairs, which is logically equivalent to making no interchanges at all.

At this point the following question arises: which of the pairwise interchanges in the list ought to actually be effected? One approach would be to only implement the k th interchange if both (1) the previous $k-1$ interchanges have been implemented, and (2) the first k interchanges all have non-negative gains $\Delta M(x^*, y^*)$. However, this strategy would fail in the case where the first few interchanges produced a (small) negative gain, i.e., worsened the goodness measure for the decomposition, but later interchanges produced larger positive gains, so that the combined impact turned out to be positive. This situation is shown graphically in Figure 6.3. Experience with the interchange algorithm has shown that such situations are not uncommon.

In this figure, the cumulative impact is negative during the first

AD-A074 911

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC METHODOLOGY FOR DESIGNING THE ARCHITECTURE OF COMP--ETC(U)
SEP 79 S L HUFF, S E MADNICK
CISR-P010-7906-12

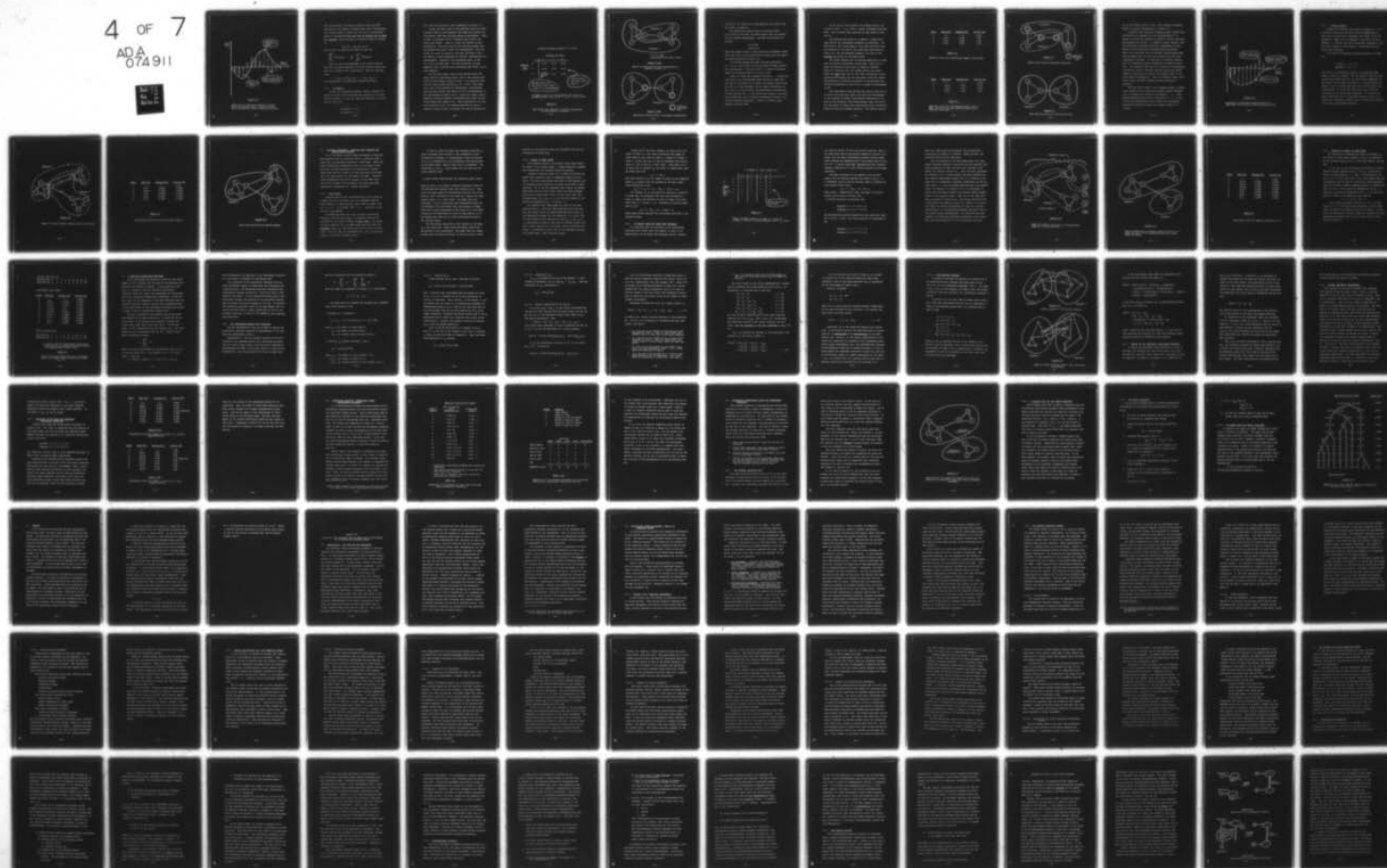
N00039-78-G-0160

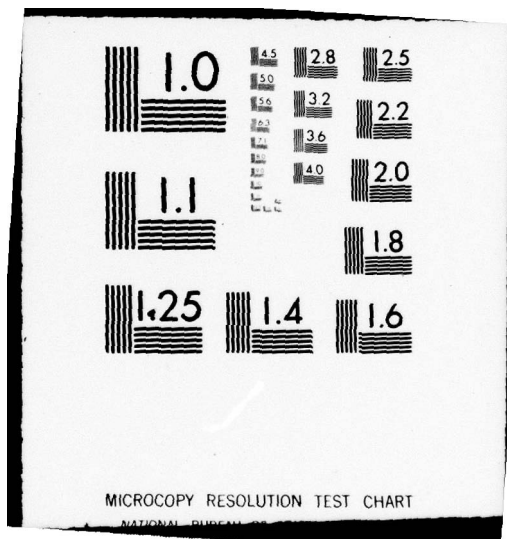
NL

UNCLASSIFIED

4 OF 7

ADA
074911





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS

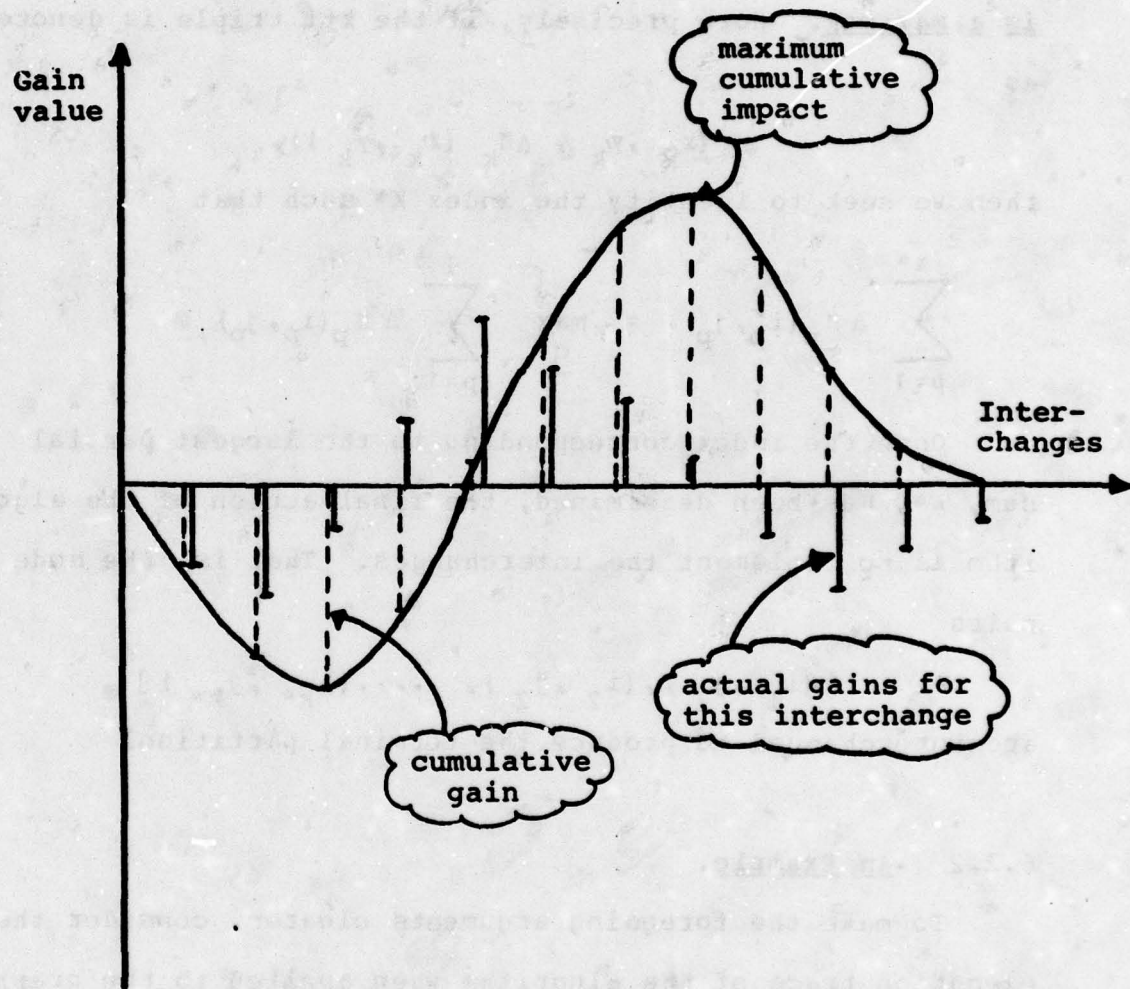


Figure 6.3

Depiction of interchange sequence in which early interchanges have a negative effect, but a net positive effect eventually accrues.

four interchanges, but becomes positive with the fifth interchange, and reaches a maximum after eight interchanges. This example makes it clear that the first k interchanges should be implemented such that the k th partial sum of gains is a maximum. More precisely, if the k th triple is denoted as

$$(x_k, y_k, \Delta M_k(x_k, y_k)),$$

then we seek to identify the index k^* such that

$$\sum_{p=1}^{k^*} \Delta M_p(i_p, j_p) = \max_q \sum_{p=1}^q \Delta M_p(i_p, j_p)$$

Once the index corresponding to the largest partial sum, k^* , has been determined, the final action of the algorithm is to implement the interchanges. That is, the node pairs

$$[(i_1, j_1), (i_2, j_2), \dots, (i_{k^*}, j_{k^*})]$$

are interchanged to produce the terminal partition.

6.2.2 An Example.

To make the foregoing arguments clearer, consider the execution trace of the algorithm when applied to the graph shown in Figure 6.1, with the following arbitrarily selected initial partition:

partition 1: 1 2 6

partition 2: 3 4 5

This initial partition is that illustrated in Figure 6.2.

During the first cycle of the algorithm there are three eligible nodes in each subgraph (all nodes are eligible for interchange; none have yet been marked as ineligible). Thus we want to calculate 3×3 or 9 possible gain values, and select the largest. Table 6.1 gives the results of this calculation. From this table we see that the largest gain is achieved when nodes 5 and 1 are interchanged. Thus the triple (5,1,.66) is placed on the list, and nodes 5 and 1 are marked as being ineligible for further consideration for interchanges. Logically, the interchange shown in Figure 6.4(a) has been made. The above procedure is then repeated, with a reduced graph containing four nodes, two in each subgraph.

For the next cycle, since we have decided after the first cycle to interchange nodes 5 and 1, the starting partition is that shown in Figure 6.4(b). The objective function for this initial partition is $M' = 0.29$. Only nodes 2,3,4, and 6 are eligible for interchange. Calculations show that the largest gain value is -0.64 , corresponding to the interchange of nodes 4 and 2. Since the largest gain is negative, the best possible interchange still makes the partition worse with respect to M . This observation is in line with intuition, since the starting partition at cycle 6 (shown in Figure 6.4(b)) is clearly the overall optimal par-

Original Goodness Measure $M = -0.367$

		Subgraph #2 nodes		
		1	2	6
Subgraph #1 nodes	3	-0.23	-0.25	0.011
	4	-0.30	-0.19	-0.02
	5	0.65	-0.21	-0.02

Optimal gain value ΔM^* occurs when nodes 1 and 5 are interchanged

Table entries are gain values, $\Delta M(i,j)$.

If nodes 1 and 5 are interchanged, the resulting goodness measure will be $M' = -0.367 + 0.65 = 0.283$.

Table 6.1

Gain matrix for deciding on initial interchange of graph shown in Figure 6.2

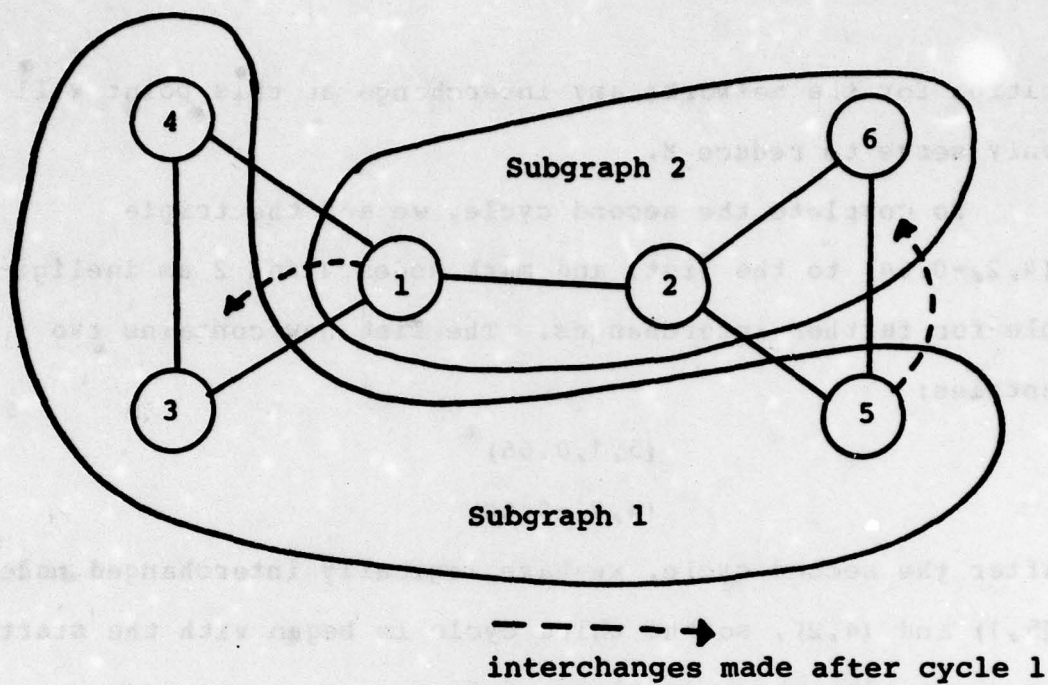


Figure 6.4(a)

Effect of first pair of node interchanges on graph of Figure 6.2.

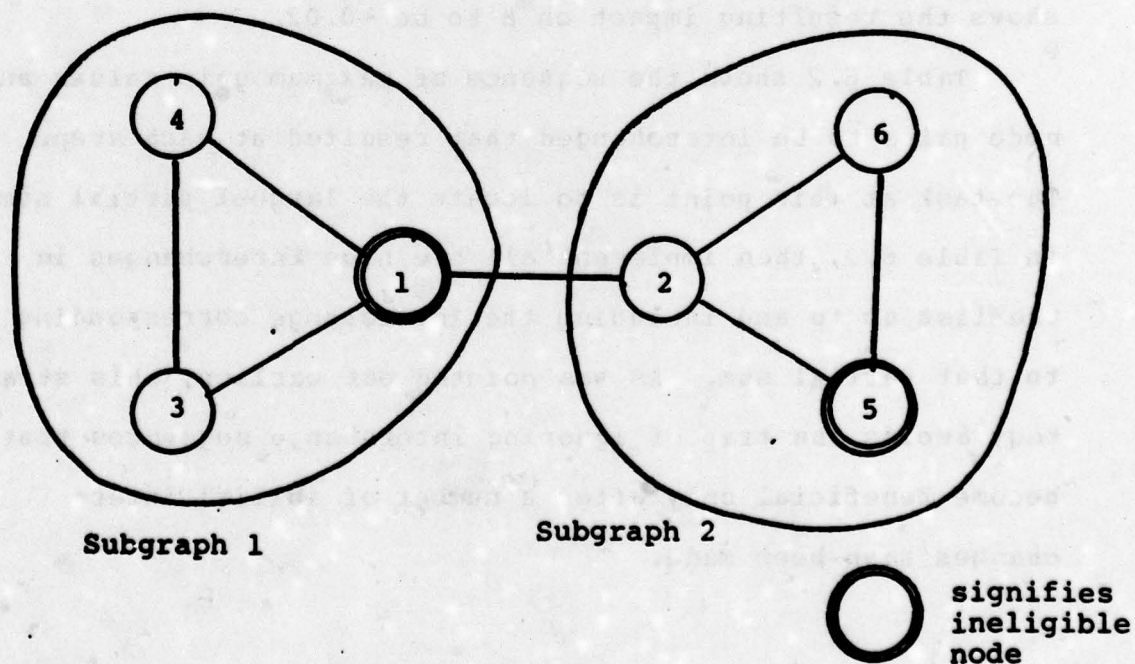


Figure 6.4(b)

Resulting situation after interchanges implemented.

tition for the network; any interchange at this point will only serve to reduce M .

To complete the second cycle, we add the triple $(4,2,-0.64)$ to the list, and mark nodes 4 and 2 as ineligible for further interchanges. The list now contains two entries:

$(5,1,0.66)$

$(4,2,-0.64)$

After the second cycle, we have logically interchanged nodes $(5,1)$ and $(4,2)$, so the third cycle is begun with the starting partition shown in Figure 6.5.

The initial measure for this starting partition is $M' = -0.35$. Since there is only one pair of nodes eligible for interchanging - node pair $(3,6)$ - a single calculation shows the resulting impact on M to be -0.02 .

Table 6.2 shows the sequence of maximum gain values and node pairs to be interchanged that resulted at each step. The task at this point is to locate the largest partial sum in Table 6.2, then implement all the node interchanges in the list up to and including the interchange corresponding to that partial sum. As was pointed out earlier, this strategy avoids the trap of ignoring interchange sequences that become beneficial only after a number of initial interchanges have been made.

In the case of this example, the largest partial sum occurs at step 1 - i.e., after a single interchange has been made. Thus the best final partition is that shown in Figure 6.6.

The question now arises as to whether a single such pass through the interchange procedure is sufficient. Put differently, what would happen if the final partition that was generated at the end of the first pass (that shown in Figure 6.6, for the foregoing example), was used as the starting partition for the second pass?

The second question may be answered empirically in this case by actually performing a second pass. When this is done, the execution trace shown in Table 6.3 results. Table 6.3 indicates that the largest partial sum occurs after the final step, and is 0.00. This corresponds to completely interchanging all the node pairs. Since the resulting partition is logically equivalent to the starting one, the best alternative in this case is to make no interchanges at all.

This phenomenon occurs because the initial partition is in fact a very good one; hence, the first few interchanges in the sequence of optimal interchanges substantially worsens the partitioning, while interchanges toward the end of the list begin to improve the partitioning again, eventually ending up with the original partition. The largest partial

<u>Cycle</u>	<u>Node pair</u>	<u>Maximum Gain</u>	<u>Partial sum</u>
0	---	---	0.00
1	(5,1)	0.66	0.66
2	(4,2)	-0.64	0.02
3	(3,6)	-0.02	0.00

Table 6.2

Execution trace for interchange example (first pass)

<u>Cycle</u>	<u>Node pair</u>	<u>Maximum Gain</u>	<u>Partial Sum</u>
0	---	---	0.00
1	(4,2)	-0.64	-0.64
2	(3,6)	-0.01	-0.65
3	(5,1)	0.65	0.00

Table 6.3

Execution trace for interchange example (second pass, starting with best identified partition from the first pass).

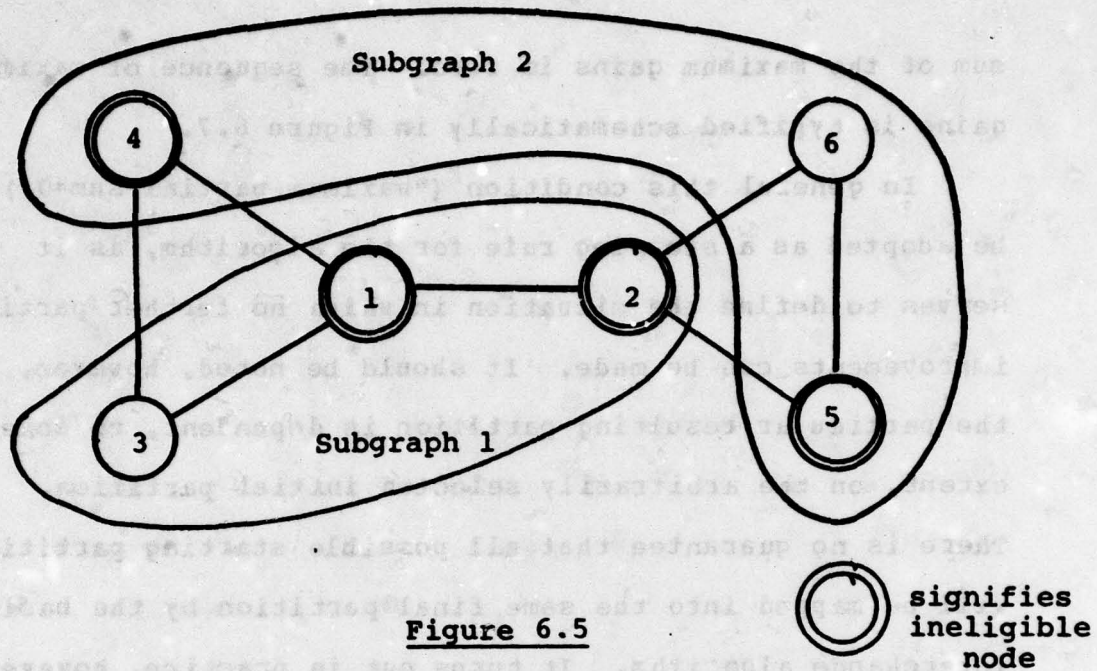


Figure 6.5

Result after second interchange completed.

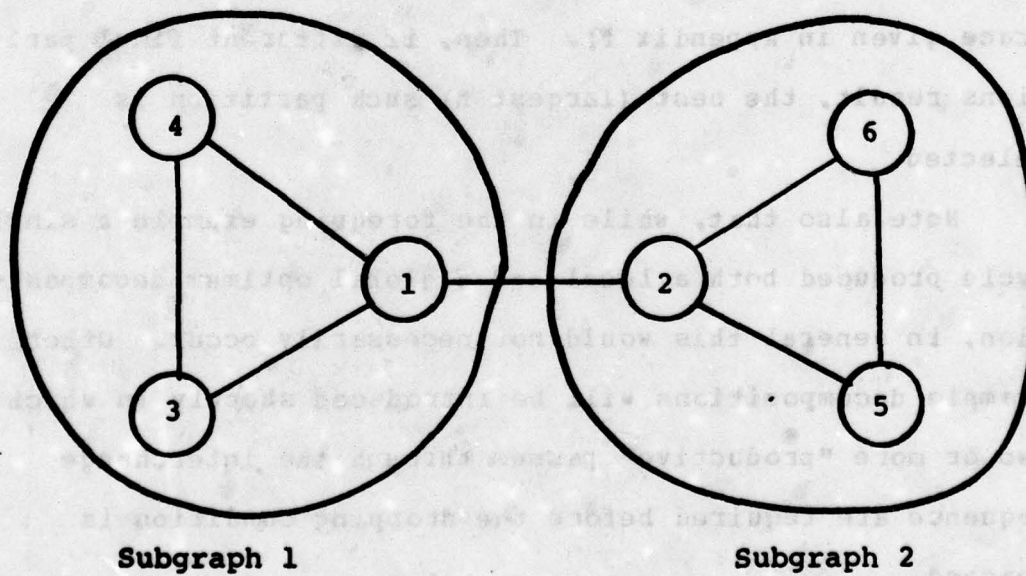


Figure 6.6

Best obtained partition during first pass

sum of the maximum gains is zero. The sequence of maximum gains is typified schematically in Figure 6.7.

In general this condition ("maximum partial sum=0") may be adopted as a stopping rule for the algorithm, as it serves to define the situation in which no further partition improvements can be made. It should be noted, however, that the particular resulting partition is dependent, to some extent, on the arbitrarily selected initial partition. There is no guarantee that all possible starting partitions will be mapped into the same final partition by the basic interchange algorithm. It turns out in practice, however, that the algorithm is quite insensitive to starting partition. The approach taken here is to execute the interchange algorithm on various different starting partitions (three different partitions are used in the example shown in the trace given in Appendix F). Then, if different final partitions result, the best (largest M) such partition is selected.

Note also that, while in the foregoing example a single cycle produced both a local and a global optimum decomposition, in general this would not necessarily occur. Other example decompositions will be introduced shortly in which two or more "productive" passes through the interchange sequence are required before the stopping condition is reached.

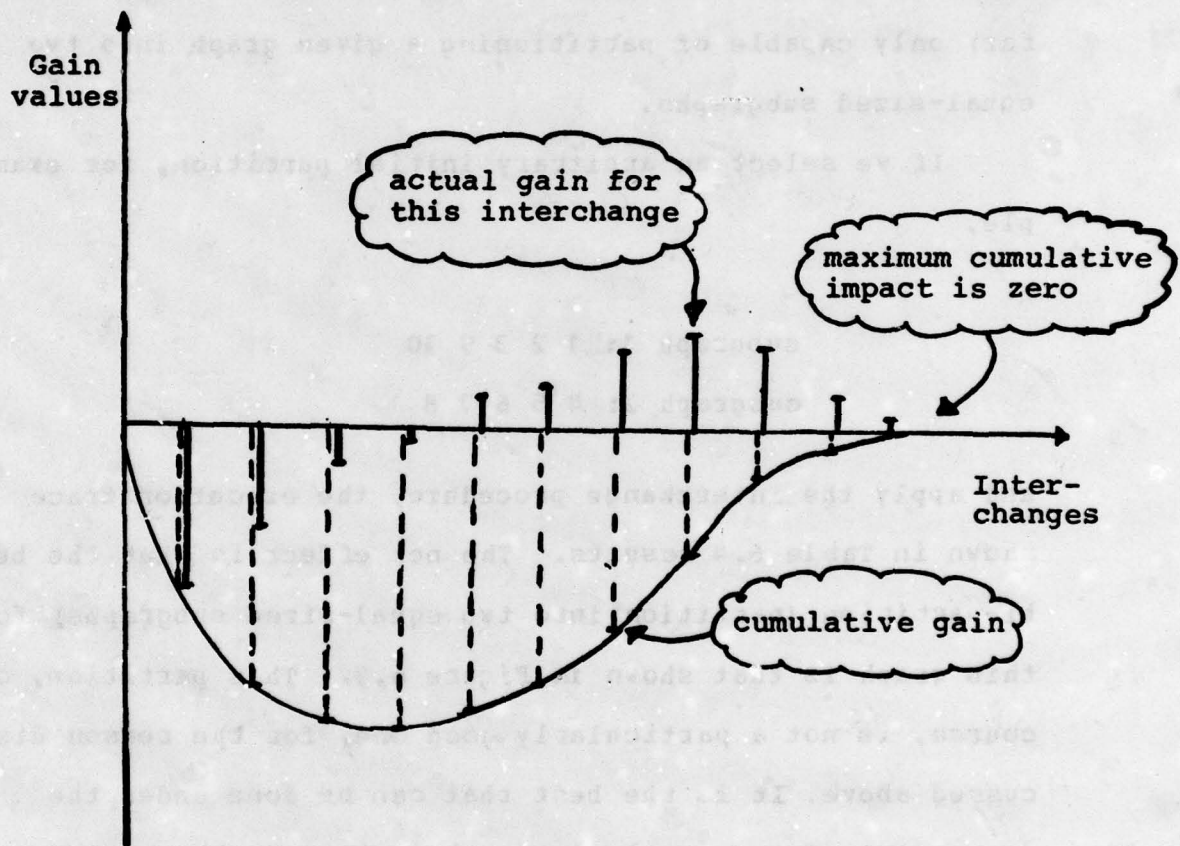


Figure 6.7

Depiction of interchange sequence wherein no improvement can be made in the starting partition.

6.2.3 A Second Example.

Consider the weighted graph shown in Figure 6.8. It is intuitively clear that the best overall decomposition of this graph contains three subgraphs, with nodes 1,2,3,4 in subgraph 1, nodes 5,6,7 in subgraph 2, and nodes 8,9,10 in subgraph 3. However, the basic interchange technique is (so far) only capable of partitioning a given graph into two equal-sized subgraphs.

If we select an arbitrary initial partition, for example,

subgraph 1: 1 2 3 9 10

subgraph 2: 4 5 6 7 8

and apply the interchange procedure, the execution trace shown in Table 6.4 results. The net effect is that the best bi-partition (partition into two equal-sized subgraphs) for this graph is that shown in Figure 6.9. This partition, of course, is not a particularly good one, for the reason discussed above. It is the best that can be done under the constraint of two equal-sized subgraphs. In the next section a technique will be introduced that will allow us to relax this constraint, thereby greatly extending the utility of the basic interchange algorithm.

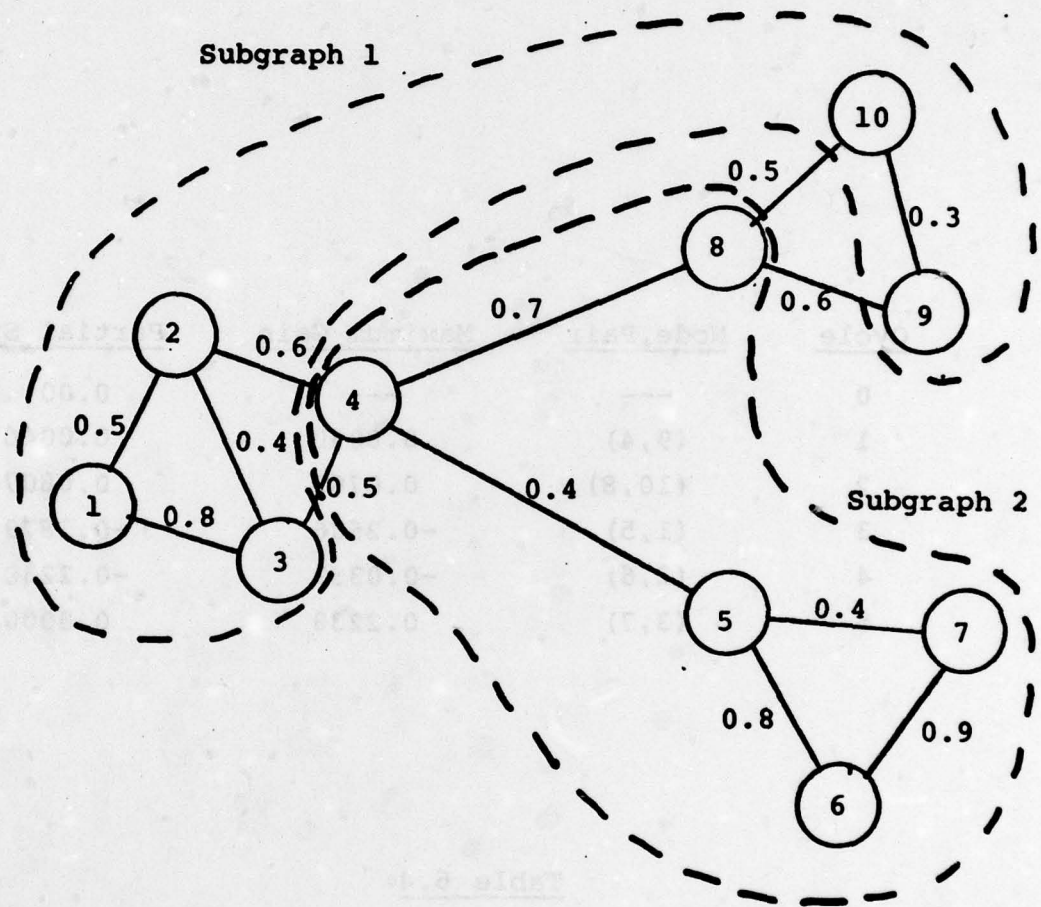
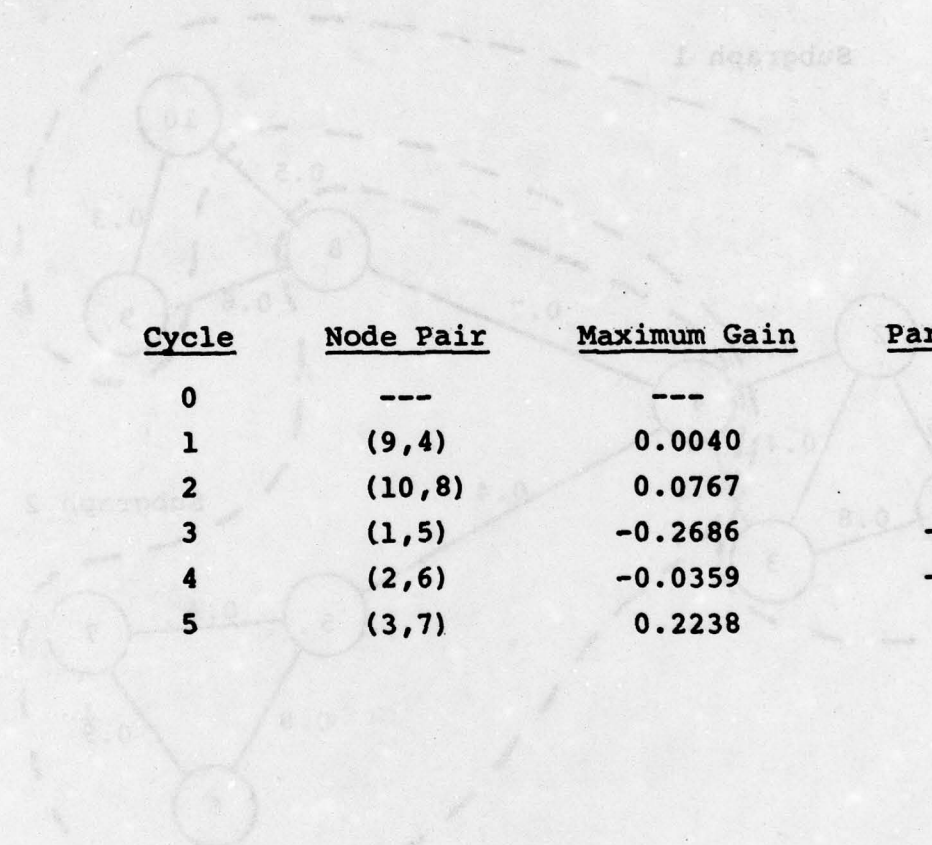


Figure 6.8

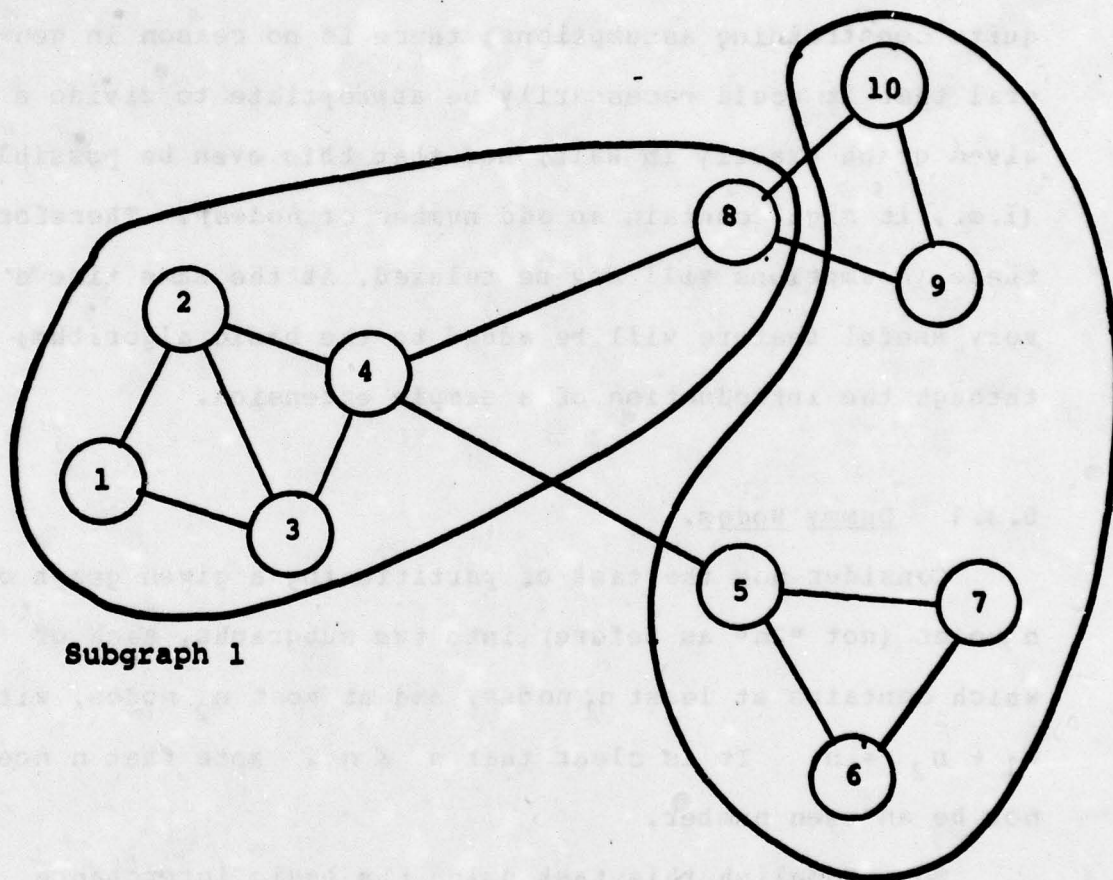
Graph for second example, showing initial partition.



<u>Cycle</u>	<u>Node Pair</u>	<u>Maximum Gain</u>	<u>Partial Sum</u>
0	---	---	0.00
1	(9,4)	0.0040	0.0040
2	(10,8)	0.0767	0.0807
3	(1,5)	-0.2686	-0.1879
4	(2,6)	-0.0359	-0.2238
5	(3,7)	0.2238	0.0000

Table 6.4

Interchange execution trace for second example.



Subgraph 1

Figure 6.9

Best final partition for second example.

6.3 ALGORITHM EXTENSIONS - PARTITION SIZE BOUNDING AND UNEQUAL-SIZED SUBSETS.

Up to this point a target graph containing $2n$ nodes has been assumed, with the objective being to subdivide such a graph into two subgraphs containing n nodes each. These are quite constraining assumptions; there is no reason in general that it would necessarily be appropriate to divide a given graph exactly in half, nor that this even be possible (i.e., it might contain an odd number of nodes). Therefore these assumptions will now be relaxed. At the same time a very useful feature will be added to the basic algorithm, through the introduction of a simple extension.

6.3.1 Dummy Nodes.

Consider now the task of partitioning a given graph of n nodes (not " $2n$ " as before) into two subgraphs, each of which contains at least n_1 nodes, and at most n_2 nodes, with $n_1 + n_2 = n$. It is clear that $n \leq n$. Note that n need not be an even number.

To accomplish this task using the basic interchange algorithm, the original graph G is augmented through the introduction of a certain number of dummy nodes. The original graph together with the dummy nodes will be termed the augmented graph, G_a . The dummy nodes have no links to other nodes - that is, they are represented by a row and column of zeroes in the graph adjacency matrix.

In order to make the dummy node technique effective, a small adjustment must be made to the calculation of the decomposition goodness, M . Specifically, M must be defined so as to be unaffected by the introduction and distribution of the dummy nodes. This is quite easy to accomplish. The individual S_i and C_{ij} terms making up M all have the following general form:

$$f_1 (\text{link weight sum}) * (\text{average link weight}) / f_2 (\text{node count}),$$

where f_1 and f_2 are slightly different functional forms for the strength and coupling terms (see Section 6.4.1). Now, since the dummy nodes are disconnected from the rest of the network, their inclusion in the graph only impacts the denominator terms, i.e., node counts. By adding the minor adjustment to the M calculation that disconnected nodes not be included in the node count term, the presence of dummy nodes can be made transparent to the value of M . This modification to the definition of M has no other effect, as it is assumed that there are no "real" disconnected nodes in the original graph.

The interchange algorithm is then applied to the graph G_a in the usual way. Dummy nodes and regular nodes both participate in the interchange. The dummy nodes are simply dropped from the final partition, to yield to yield a decom-

position of the original graph into subgraphs that are not necessarily of equal size.

6.3.2 Choice of Dummy Nodes.

The question arises as to how many dummy nodes should be added to the original graph. A simple empirical argument will demonstrate the reasoning behind the answer.

Consider a starting graph G of 5 nodes, and assume one dummy node is added to give an augmented graph G_a of six nodes. The basic interchange algorithm is then applied, and the resulting final partition will have three nodes in each subgraph. One of the two subgraphs will contain the single dummy node and two "real" nodes, while the other subgraph will contain three "real" nodes. In this case, then, the original graph is of size $n = 5$, and the size bounds on the resulting subgraphs are $n_1 = 2$ and $n_2 = 3$.

Now suppose three dummy nodes are added to the same starting graph G . Then the final subgraphs will each contain four nodes. Since the dummy nodes may end up split in any combination between the two subgraphs, it is clear that the size bounds on the subgraphs are now $n_1 = 1$ and $n_2 = 4$.

It would make no sense in this case to consider adding more than 3 dummy nodes (i.e., 5 or more), since a "trivial" partition - a partition in which one of the subgraphs contained only dummy nodes - might otherwise result.

Repetition of the above argument for other sizes of G leads to Table 6.5. This table illustrates the number of dummy nodes n_d that must be added to a graph of n nodes in order to insure that each of the two subgraphs will have at least n_1 , and at most n_2 , "real" nodes. From Table 6.5 it is clear that in general n_d , the number of dummy nodes, must be chosen such that

$$n_d = n - 2*n_1 .$$

With this choice of n_d , the number of nodes in the augmented graph G_a is always even (as required by the basic interchange algorithm), since

$$n_t = n + n_d = n + (n - 2*n_1) = 2*(n - n_1) .$$

For example, if it was desired to partition a graph of 43 nodes into two subgraphs such that each contained at least 10 nodes, and therefore at most 33 nodes, this would imply that $n_1 = 10$ and $n = 43$. Therefore it would be necessary to add

$$n_d = n - 2*n_1 = 43 - 2*(10) = 23$$

dummy nodes before applying the interchange algorithm to the augmented graph.

6.3.3 An Example Using the Dummy Node Technique.

To illustrate both the operation of the interchange algorithm while dummy nodes are present, as well as the effectiveness of the dummy node technique itself, consider

n = number of "real" nodes in G

		5	6	7	8	...
n_1	1	3	4	5	6	
	2	1	2	3	4	
lower bound on partition size	3	--	0	1	2	
	4	--	--	--	0	
	5	--	--	--	--	
	.					
	.					
	.					

Table entries are n_d values.

Table 6.5

Number of dummy nodes to be added to a graph of n nodes to insure a minimum subgraph size of n_1 nodes.

the graph of Figure 6.8 from the previous section. This is the same graph used in the earlier example of Section 6.2.3. Recall that the basic interchange algorithm without dummy nodes produced the decomposition of this graph shown in Figure 6.9 - a clearly suboptimal decomposition that resulted primarily because of the exact halving property of the basic algorithm.

The basic algorithm is now applied to the current graph, with the minimum subgraph size taken to be $n_1 = 3$. An augmented graph G_a must be defined, where G_a consists of the original graph G plus

$$n_d = n - 2*n_1 = 10 - 2*3 = 4$$

dummy nodes. Schematically, then, the graph to be partitioned is that shown in Figure 6.10.

An initial partition is selected, say,

subgraph 1: 1 2 3 9 10 11 12

subgraph 2: 4 5 6 7 8 13 14

The decomposition goodness measure for this partition turns out to be $M' = -0.29$. The final partition is determined to be:

subgraph 1: 1 2 3 4 8 9 10

subgraph 2: 5 6 7 11 12 13 14

After the dummy nodes are discarded, the decomposition illustrated in Figure 6.11 results. Table 6.6 gives the execution trace for this partition.

With the addition of the four dummy nodes, the interchange algorithm produced a partition with the smallest subgraph containing at least (in this case, exactly) three nodes, the selected value of n_1 . Also, the final partitioning (Figure 6.11) is clearly better than that obtained earlier without dummy nodes (Figure 6.9). It is better in the sense that it represents a "natural" subdivision of the original graph. If the interchange algorithm was to be applied again, to subgraph 2 in Figure 6.11, it is reasonable to expect that the result would be the optimal global decomposition of the original graph. (This is in fact what happens, as shown in Section 6.5.) This result should be contrasted with that shown in Figure 6.9, wherein it is clear that since the first partitioning is a poor one, further partitionings of the resulting subgraphs can never lead to the global optimum decomposition. As illustrated by this example, the dummy node technique contributes a very important and useful extension to the interchange algorithm.

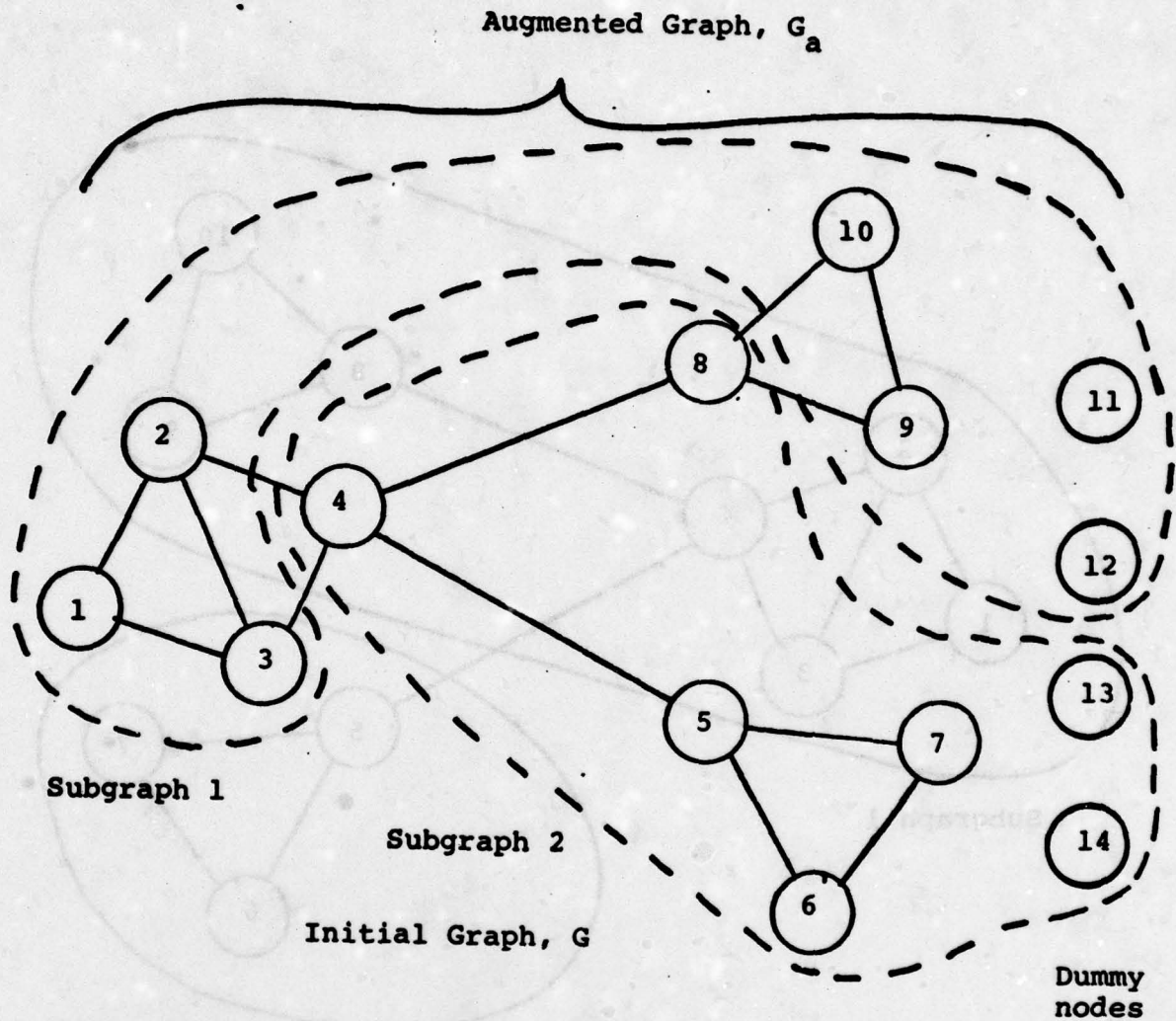


Figure 6.10

Graph for example of Section 6.3 showing dummy nodes and initial partition.

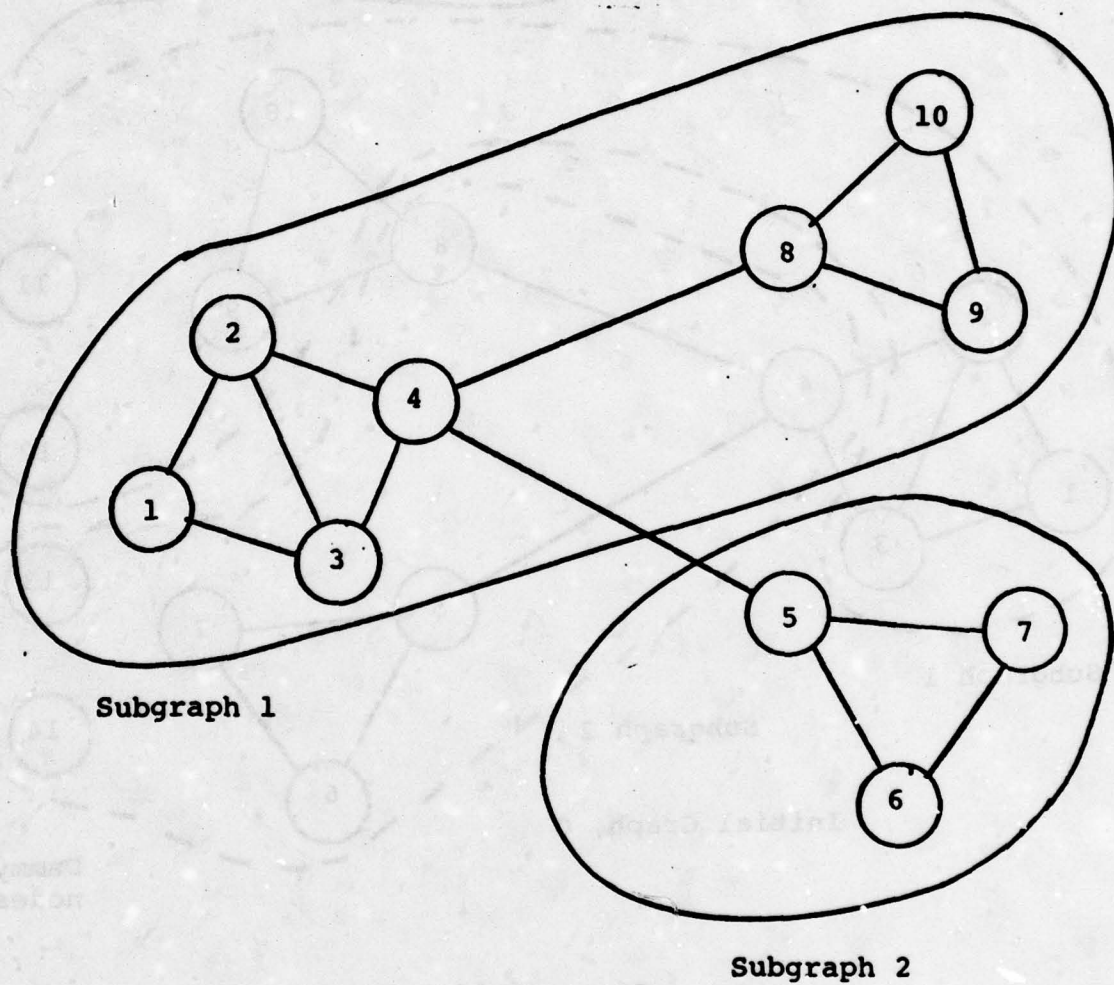


Figure 6.11

Final decomposition of example graph of Section 6.3.3 after one pass of the interchange algorithm (dummy nodes not shown).

<u>Cycle</u>	<u>Node pair</u>	<u>Maximum gain</u>	<u>Partial sum</u>
0	---	---	0.000
1	(12,8)	0.0876	0.0876
2	(11,4)	0.2284	0.3158
3	(10,13)	-0.2639	0.0519
4	(9,14)	0.0286	0.0804
5	(1,5)	-0.2686	-0.1881
6	(2,6)	-0.0359	-0.2239
7	(3,7)	0.2239	0.0000

Table 6.6

Interchange trace for example of Section 6.3.3.

6.3.4 Choosing the Number of Dummy Nodes.

Two alternative considerations arise in choosing n_d , the number of dummy nodes: whether or not it is desired to set the minimum subgraph size at some level greater than one node.

If the user of the interchange algorithm has in mind a specific minimum subgraph size n_1 , then n_d is determined from the relationship $n_d = n - 2*n_1$, as discussed in Section 6.3.1. On the other hand, it may often happen that the investigator does not want to disallow any small subgraphs. Since the smallest possible subgraph consists of exactly one node, in such a case it would be necessary to set $n_1 = 1$. From this, n_d would be determined to be $n - 2$, so the total number of nodes in the augmented graph would be

$$n = n + (n-2) = 2*n - 2.$$

In the previous example, Section 6.7, if n_1 had been taken to be 1, it would have been necessary to add 8 dummy nodes, to produce an augmented graph of 18 nodes. In this case, the interchange algorithm would have produced the same partitioning obtained earlier, as is seen from the trace given in Table 6.7.

- Initial Partition is:

Partition #1: 1 2 3 9 10 11 12 15 16
 Partition #2: 4 5 6 7 8 13 14 17 18

Interchange Trace Table:

<u>Cycle</u>	<u>Node pair</u>	<u>Maximum gain</u>	<u>Partial sum</u>
0	---	---	0.0000
1	(16,8)	0.0876	0.0875
2	(15,4)	0.2284	0.3158
3	(11,13)	0.00	0.3158
4	(12,14)	0.00	0.3158
5	(10,17)	-0.2639	0.0519
6	(9,18)	0.0286	0.0804
7	(1,5)	-0.2686	-0.1881
8	(2,6)	-0.0359	-0.2239
9	(3,7)	0.2283	0.0000

Final partition is:*

Partition #1: 1 2 3 4 8 9 10 13 14
 Partition #2: 5 6 7 11 12 15 16 17 18

* A second round of interchange calculations using this as the starting partition results in a maximum partial sum = 0.00.

Table 6.7

Initial and final partitions, and interchange trace, for example graph of Section 3.3 with $n_1 = 1$.

6.4 A SIMPLIFIED INTERCHANGE ALGORITHM.

One of the reasons originally put forth for the development of this algorithm was the need for an efficient hierarchical partitioning technique. It was pointed out earlier, for instance, that the iterative partitioning technique studied by Andreu was interesting but impractical, due to its inefficient operational properties. In its present form, the new algorithm is much more efficient than the iterative approach. However, certain minor simplifications in the calculations of the interchange technique may be made in order to make it considerably faster still.

The main bottleneck in the present form of the algorithm centers on the gain calculation. If there are $2n$ eligible nodes, n^2 calculations are required to locate the largest gain value during the first cycle. In the next cycle, $(n-1)^2$ calculations will be required, etc. The total number of gain calculations will be, for a network originally containing $2n$ nodes,

$$T = \sum_{k=1}^n k^2$$

This is a fairly large number of calculations steps for any non-trivial values of n . For example, if $2n = 200$ nodes, then $n = 100$, and

$$T = 100*100 + 99*99 + \dots + 2*2 + 1*1 = 338,350.$$

Some efficiencies are necessary if the interchange algorithm is to be useful in problems of significant size.

Now, operation of the interchange technique could be made considerably faster by simplifying the interchange calculation itself - the elementary node pair interchange and corresponding measure calculation - since this step is executed so many times. In this section we develop such a simplification through incorporation of an approximate measure gain criterion, and give an example of its use. In practice the simplified algorithm, based on the approximate gain criterion has been found to behave essentially as well as the original algorithm, in terms of its capability for producing good partitions.

6.4.1 The Approximate Measure Gain Criterion.

The key simplification that can be made to improve the algorithm's efficiency involves an approximation to the measure gain calculation, $\Delta M(i, j)$.

Calculation of $\Delta M(i, j)$ basically consists of the calculation of the goodness measure for a particular bi-partition of a given graph, namely, the bi-partition obtained by interchanging the nodes x and y with respect to some initial bi-partition. Now, the definition of M for any graph that

has been decomposed into k subgraphs is given as

$$M = \sum_{i=1}^k S_i - \sum_{i=1}^{k-1} \sum_{j=i+1}^k C_{ij}$$

When the number of subgraphs k is equal to 2, this becomes

$$M = S_1 + S_2 - C_{12}.$$

The definitions of strength and coupling for a weighted graph (from Chapter 4) are:

1. strength S_i of subgraph i :

$$S_i = [L_i - (n_i - 1)] / [(n_i)(n_i - 1)/2 - (n_i - 1)] \bar{w}_i$$

where L_i = the number of links within i

n_i = the number of nodes within i

\bar{w}_i = the average weight on links within i .

2. coupling C_{ij} between subgraphs i and j :

$$C_{ij} = [L_{ij} / (n_i n_j)] \bar{w}_{ij}$$

where L_{ij} = the number of links between i and j

n_i, n_j = the number of nodes in i, j

\bar{w}_{ij} = the average weight on links between i and j .

6.4.1.1 Simplifying S_i .

First consider the S_i term. This may be written

$$S_i = 2/[(n_i-1) (n_i-2)] * L_i \bar{w}_i - [2/(n_i-2)] * \bar{w}_i$$

Now, a pairwise node interchange does not affect the value of n_i , so n_i is a constant as far as the calculation of $\Delta H(x,y)$ is concerned. Also, since \bar{w}_i is an average, it may be assumed that its value is not substantially affected by interchanging nodes x and y . This is the case since most of the link weights that go into the calculation of \bar{w}_i would remain unchanged. Therefore the entire second term in the S_i expression may be treated as a constant term. Since in the calculation of $\Delta H(x,y)$, this constant value will subtract out, it may be dropped henceforth.

Also, it may be noted that \bar{w}_i is defined as W_i/L_i , where W_i is the sum of the weights on the links in subgraph i , and L_i is the link count in subgraph i . Hence the effective expression for S_i becomes

$$S_i = 2/[(n_i-1) (n_i-2)] * W_i$$

6.4.1.2 Simplifying C_{ij} .

If w_{ij} is defined as the sum of the weights on links connecting subgraphs i and j , then $\bar{w}_{ij} = w_{ij}/L_{ij}$. Thus the expression for C_{ij} simplifies to

$$C_{ij} = 1/(n_i n_j) * w_{ij}.$$

6.4.1.3 Further Simplification of $\Delta M(x,y)$.

If we let S_1, S_2 , and C_{12} be the values of strength and coupling for some starting bi-partitioned graph, and let S_1^i , S_2^j , and C_{12}^i be the equivalent values after nodes x and y have been interchanged. Then

$$\Delta M(x,y) = (S_1^i + S_2^j - C_{12}^i) - (S_1 + S_2 - C_{12}).$$

Since equal-sized subgraphs are being considered, we may let $n = n_1 = n_2$, and the expression for $\Delta M(x,y)$ becomes

$$\Delta M(x,y) = 2/[n(n-1)] * (w_1^i + w_2^j - w_1 - w_2) - (1/n) * (w_{12}^i - w_{12}).$$

If we now approximate $(n-1)(n-2)$ by n^2 , we may factor out a $1/n^2$ term and get

$$\Delta M(x,y) = (1/n^2) * [2 * (w_1^i + w_2^j - w_1 - w_2) - (w_{12}^i - w_{12})].$$

Now, the interchange algorithm is eventually going to make an ordinal comparison among all the $\Delta H(x,y)$ values for all (x,y) combinations. Thus the constant $1/n^2$, which will be present as an external multiplier in each term, may be dropped from the $\Delta H(x,y)$ expression. This is so since it will be present in all such terms, and consequently serves only to scale all the values, having no net impact on their ordinal relationship.

Therefore the effective value for $\Delta H(x,y)$ reduces to

$$H(x,y) \propto 2(w'_1 + w'_2 - w_1 - w_2) - (w'_{12} - w_{12}) \quad \dots\dots(1)$$

In order to go further with the reduction of the expression for $\Delta H(x,y)$, it is necessary to introduce some new terminology. We define:

1. E_x = the sum of the weights on the external links connected to node x - that is, those links that connect node x to some node in the other subgraph;
2. I_x = the sum of the weights on the internal links connected to node x - that is, those links that connect node x to other nodes within the same subgraph;
3. O_x = the sum of the weights on the "other" links within the same subgraph as node x - i.e., those links not connected to node x ;
4. O_{xy} = the sum of the weights on the "other" links interconnecting the two subgraphs - i.e., those links not connected to either node x nor node y ;

5. w_{xy} = the weight on the link connecting node x to node y (if no such link exists, w_{xy} is assumed to be zero).

Now, the W terms in the earlier expression for $\Delta H(x,y)$ may be translated into terms involving E, I, O , and w . In particular, it is fairly easy to see that

$$W_1 = I_x + O_x \quad \text{..... (2)}$$

$$W_2 = I_y + O_y \quad \text{..... (3)}$$

$$W'_1 = E_y - w_{xy} + O_x \quad \text{..... (4)}$$

$$W'_2 = E_x - w_{xy} + O_y \quad \text{..... (5)}$$

$$W_{12} = E_x + E_y - w_{xy} + O_{xy} \quad \text{..... (6)}$$

$$W'_{12} = I_x + I_y + w_{xy} + O_{xy} \quad \text{..... (7)}$$

The only slightly confusing part in the above equations involves the role of w_{xy} . When x and y are interchanged, all associated external links become internal, and vice versa, with the exception of the link connecting x and y (if any).

Now, if equations (2) through (7) are substituted into relation (1), the latter reduces to

$$\begin{aligned} \Delta H(x,y) &\propto 2[(E_x - I_x) + (E_y - I_y) - 2w_{xy}] \\ &\quad - [(I_x - E_x) + (I_y - E_y) + 2w_{xy}] \\ &= 3[(E_x - I_x) + (E_y - I_y) - 2w_{xy}] \quad \text{..... (8)} \end{aligned}$$

It is no surprise that the "0" terms in (2) through (7) cancel out of (8), since by definition these terms represent a part of the graph structure that is unaffected by the interchange of nodes x and y.

Finally, we define

$$D_x = E_x - I_x, \text{ and}$$

$$D_y = E_y - I_y.$$

Also, for the same reason as discussed earlier (immateriality of scale factors) we drop the factor 3 in relation (8). This yields the final result,

$$\Delta M(x,y) \propto D_x + D_y - 2w_{xy} \quad \dots\dots (9)$$

Expression (9) is the simplified measure gain expression. To be strictly correct, the right-hand side of expression (9) is proportional to an approximation of the exact measure gain $\Delta M(x,y)$. As was argued, proportionality constants are immaterial for purposes of the interchange algorithm's proper functioning; also, the approximations involved in the foregoing reduction are relatively minor, so that we would expect the simplified algorithm to still do a good job in partitioning graphs in a manner appropriate to our needs - i.e., in a manner that will usually locate a very good, or optimal, partition as measured by the true value of M.

6.4.2 A Verification Exercise.

To briefly illustrate the validity of relation (9) in the previous section, consider the graph shown in Figure 6.12, with the associated subgraphs 1 and 2. Two nodes x and y , one in each subgraph, are shown; also, the weights on certain links are specified, for illustrative purposes to follow.

In Figure 6.13, the same graph is shown, after nodes x and y have been interchanged between the two subgraphs. In terms of the foregoing definitions, the following may be seen to hold:

$$w_1 = w_1 + w_2 + o_1$$

$$w_2 = w_5 + w_6 + o_2$$

$$w_{12} = w_3 + w_4 + w_7 + o_{12}$$

$$w'_1 = w_7 + o_1$$

$$w'_2 = w_3 + o_2$$

$$w'_{12} = w_1 + w_2 + w_4 + w_5 + w_6 + o_{12}$$

where o_1 and o_2 represent the sum of the weights on the "other" links in subgraphs 1 and 2 respectively (those links not attached to nodes x and y , respectively); similarly, o_{12} represents the sum of the weights on the other links that run between the two subgraphs.

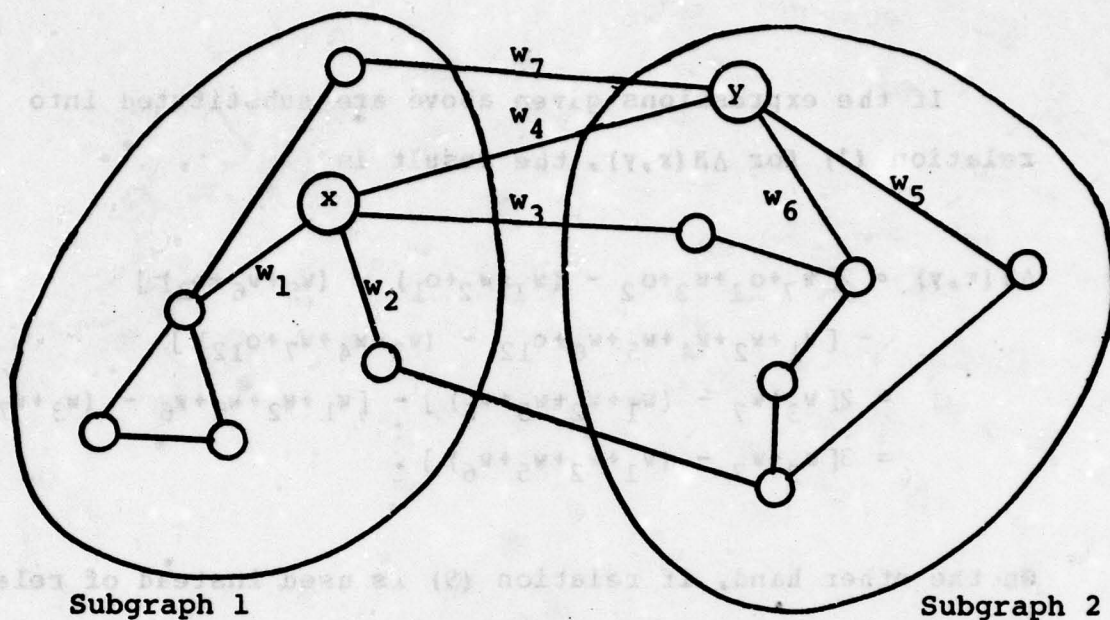


Figure 6.12

Initial graph for verification exercise

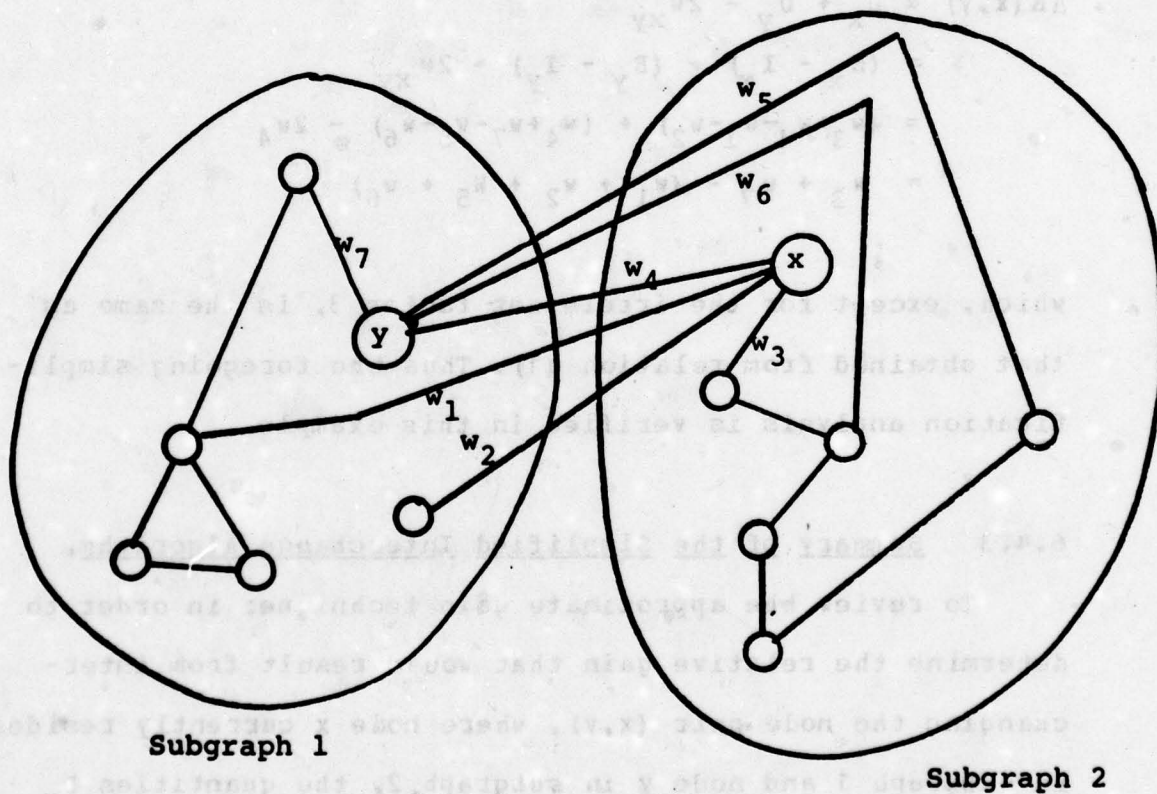


Figure 6.13

Graph of Figure 6.12 after nodes x and y have been interchanged.

If the expressions given above are substituted into relation (1) for $\Delta M(x,y)$, the result is

$$\begin{aligned}\Delta M(x,y) &\propto 2[w_7+o_1+w_3+o_2 - (w_1+w_2+o_1) - (w_5+w_6+o_2)] \\ &\quad - [w_1+w_2+w_4+w_5+w_6+o_{12} - (w_3+w_4+w_7+o_{12})] \\ &= 2[w_3+w_7 - (w_1+w_2+w_5+w_6)] - [w_1+w_2+w_5+w_6 - (w_3+w_7)] \\ &= 3[w_3+w_7 - (w_1+w_2+w_5+w_6)].\end{aligned}$$

On the other hand, if relation (9) is used instead of relation (1), the result is

$$\begin{aligned}\Delta M(x,y) &\propto D_x + D_y - 2w_{xy} \\ &= (E_x - I_x) + (E_y - I_y) - 2w_{xy} \\ &= (w_3+w_4-w_1-w_2) + (w_4+w_7-w_5-w_6) - 2w_4 \\ &= w_3 + w_7 - (w_1 + w_2 + w_5 + w_6)\end{aligned}$$

which, except for the irrelevant factor 3, is the same as that obtained from relation (1). Thus the foregoing simplification analysis is verified in this example.

6.4.3 Summary of the Simplified Interchange Algorithm.

To review the approximate gain technique: in order to determine the relative gain that would result from interchanging the node pair (x,y) , where node x currently resides in subgraph 1 and node y in subgraph 2, the quantities D_x

and D_y are calculated. In general, D_k is calculated by summing the weights on the links that extend from node k to nodes in the other subset, then subtracting the sum of the weights on the links that extend from node k to other nodes within the same subset. Once D_x and D_y have been determined, the approximate gain term $G(x,y)$ is calculated from the formula

$$G(x,y) = D_x + D_y - w_{xy}.$$

The quantity $G(x,y)$ is the approximation to the quantity $\Delta M(x,y)$ used earlier in the basic interchange algorithm.

By inspection of the algorithms' code, the estimated ratio of calculation times for $G(x,y)$ to $\Delta M(x,y)$ is on the order of 1 to 100 - i.e., the calculation time for $G(x,y)$ is approximately 1 percent of the calculation time for $\Delta M(x,y)$. This large difference is the source of the efficiency improvement that results from using the approximate gain criterion.

Once $G(x,y)$ has been calculated for every eligible node pair, the procedure continues as before (see Section 6.2.3) to locate the largest gain value, logically interchange the corresponding nodes, mark those nodes as ineligible for further consideration, then repeat with one fewer node in each subset. The final calculation of the maximum partial

sum of gain values, which determines the terminal partition, is also carried out exactly as before.

6.4.4 Further Efficiency Improvements.

The calculation of $G(x,y)$ is almost fully uncoupled with respect to the two nodes x and y . That is, with the exception of the w_{xy} term, $G(x,y)$ involves D_x and D_y , which can be calculated individually for nodes in subgraphs 1 and 2, respectively. This gives us a means of improving the speed of the interchange algorithm still further. The basic idea is as follows. First, all the D_x and D_y terms are sorted into descending order. Then $G(x,y)$ is calculated for each pair (x,y) such that x and y are within some arbitrary count δ from the top of the D_x and D_y lists, respectively. The largest such value of $G(x,y)$ is then selected, thereby identifying the next pair of nodes to be interchanged. This technique avoids a global search over all (x,y) pairs at the cost of possibly missing the largest gain value. However, the likelihood of missing the overall largest value of $G(x,y)$ may be made as small as desired by setting the value of δ high enough. The extra work involved in this approach consists of sorting two lists of n elements each, while the saved calculations involve a proportion $(n - \delta/n)$ of the n potential calculations. The extra improvement in algorithm efficiency mounts rapidly with n . Judging by a few cases examined in detail, δ need not be very large to insure

locating the largest $G(x,y)$ value - e.g., $\delta = 10$ percent seems to be more than sufficient in the cases examined. Additional studies may suggest ways in which estimate on the basis of w_{xy} , D_x and D_y values.

6.4.5 Comparison of the Basic and Simplified Interchange Algorithms.

Consider once again the 10-node graph introduced in Figure 6.8(a). This graph is augmented with the addition of four dummy nodes, and both the basic and simplified interchange algorithms are applied to it, using the starting partition given below:

subgraph 1: 1 3 5 7 9 11 13

subgraph 2: 2 4 6 8 10 12 14

The traces for the first pass of each algorithm are shown in Table 6.8(a) and Table 6.8(b) respectively.

Basically, both versions of the algorithm produced the same final partition at the end of the first pass, following essentially the same locus of interchanges. When a second pass was taken through each version of the algorithm, the results were again identical and the loci of interchanges were also nearly identical. Of course, in all cases the actual numerical values of gain were rather different between the two versions, since the gain calculation differ-

<u>Cycle</u>	<u>Node pair</u>	<u>Maximum Gain</u>	<u>Partial Sum</u>
0	---	---	0.000
1	(9,6)	0.338	0.338
2	(13,2)	0.085	0.423
3	(11,4)	0.139	0.562 (optimal)
4	(1,12)	-0.214	0.348
5	(3,14)	-0.014	0.334
6	(7,8)	-0.240	0.094
7	(5,10)	-0.094	0.000

Table 6.8 (a)

Interchange trace for example of Section 6.4, without simplifications.

<u>Cycle</u>	<u>Node Pair</u>	<u>Maximum Gain</u>	<u>Partial Sum</u>
0	---	---	0.00
1	(9,6)	2.60	2.60
2	(11,2)	0.30	2.90
3	(13,4)	0.80	3.70 (optimal)
4	(1,12)	-1.30	2.40
5	(3,14)	-0.10	2.30
6	(7,8)	-1.70	0.60
7	(5,10)	-0.60	0.00

Table 6.8(b)

Interchange trace for example of Section 6.4, with simplifications.

ences are the essence of the difference between the two algorithms. Also, the order in which dummy nodes were handled varied somewhat due to minor implementation differences. This has no impact on the interchanges of "real" nodes, which is the important issue. The main invariant factor is the ordinal relationship among the gain values in both cases. Inspection of Tables 6.8 (a) and (b) shows that this invariance of ordinality is indeed maintained for this example.

6.5 COMPARATIVE ANALYSIS - INTERCHANGE VERSUS HIERARCHICAL CLUSTERING.

It is enlightening to compare the effectiveness of the interchange algorithm against the four hierarchical clustering algorithms studied earlier. Each of the graphs used in the comparative analysis of the clustering algorithms (Section 5.3.2) was also decomposed using the interchange algorithm. The results are summarized in Table 6.9. The net result is that in all but one case the interchange algorithm produced at least as good a decomposition as the best clustering routine, and in five of the 13 cases produced a better result than the best clustering routine. In the one case where interchange failed to produce a result as good as the best clustering routine, it found one almost as good. (16)

Weighed against this superior performance is the fact that interchange is more costly to use in terms of computer, and to some extent human, resources. For instance, interchange required approximately 9 CPU seconds to decompose the 40-node graph (case 1 in Table 5.1, Chapter 5), whereas the clustering routines each required no more than about 3 seconds. Also, the current software used to execute the interchange routine may be used in an "exploratory" fashion (the main parameters being the minimum subgraph size, and choice

(16) the result produced by interchange in this case exceeded the results of the other three clustering algorithms.

Objective function (M) values

<u>Graph ID number</u>	<u>Best clustering result (algorithm no.)</u>	<u>Interchange result</u>
1	0.098 (2)	0.123 *
2	0.10 (3)	0.157 *
3	0.08 (4)	0.107 *
4	0.24 (1,2,3)	0.24 =
5	0.11 (2)	0.125 *
6	0.061 (4)	0.074 *
7	0.085 (4)	0.075 +
8	0.41 (3,4)	0.94 *
9	0.39 (1,3)	0.39 =
10	0.48 (1,2,3,4)	0.48 =
11	0.28 (1,2,3,4)	0.28 =
12	0.05 (1,2,3,4)	0.05 =
13	0.19 (4)	0.19 =

* cases where interchange exceeded best clustering algorithm;

+ case where interchange failed to do as well as best clustering algorithm;

= cases where interchange and best clustering algorithm did equally well.

Table 6.9

Comparison of interchange and best result obtained using hierarchical clustering.

<u>Weight</u>	<u>Category</u>
5	Clear winner
4	Tied for first
3	Second or tied for second
2	Third or tied for third
1	Fourth or tied for fourth
0	Fifth or tied for fifth

	<u>Algorithm</u>				
	<u>HIER1</u>	<u>HIER2</u>	<u>HIER3</u>	<u>HIER4</u>	<u>INTERCHANGE</u>
Clear winner	0	0	0	1	6
Tied for first	5	4	5	4	6
2nd or tied	1	4	3	4	1
3rd or tied	3	4	4	3	0
4th or tied	3	1	1	0	0
5th	1	0	0	1	0
Composite score	32	37	38	39	57

Table 6.10

Comparison of the weighted performance of interchange and the four hierarchical clustering routines.

of next subgraph to be partitioned). Therefore the user of the package must spend somewhat more time using interchange to locate an optimal result for a given graph. Alternatively, an automatic "governor" may be used to guide the execution of interchange, saving the user time, but removing the opportunity for exploring for potentially superior partitions.

If we re-do the weighted comparison shown earlier in Tables 5.2 and 5.3, using now a weight of 5 for "clear winner," 4 for "tied for first," etc., the new results with interchange included are as shown in Table 6.10. Interchange earns a score of 57, while the clustering techniques score from 32 to 39. All in all, then, the interchange algorithm is seen to be a powerful, if somewhat less efficient, technique for SDM graph decomposition. The interchange algorithm has been incorporated into the current SDM analysis package, and its use is described briefly in Appendix D as part of the documentation of the SDM analysis package.

6.6 HIERARCHICAL PARTITIONING USING THE INTERCHANGE TECHNIQUE.

The original objective in developing this partitioning algorithm was to devise a means of decomposing a graph into subgraphs so as to locate the best overall decomposition as determined by the value of M . At this point we have demonstrated an efficient algorithm that is capable of dividing a given graph into two subgraphs, with the option of controlling the size of the subgraphs. All that is required to meet the original objective is some sort of "master" algorithm, within which the interchange algorithm may be imbedded, which will perform the following tasks:

1. keep track of the original graph and the list of subgraphs
2. decide which subgraph on the list should be partitioned next (using the interchange routine)
3. generate initial partitions to accompany each call to the interchange routine;
4. monitor the results of the interchange execution for the occurrence of the stopping condition (maximum partial sum of gains = 0) for any given starting subgraph and starting partition.

6.6.1 The Subgraph Selection Rule.

The only really non-trivial issue as far as the master algorithm is concerned involves the nature of the decision rule to be used to select the next subgraph for partitioning. Consider for illustrative purposes the initial 10-node

graph given earlier (see Figure 6.8(a)). As was shown in the previous section, the best general bi-partitioning of this graph is that illustrated in Figure 6.14 below. Now we wish to determine which of the two subgraphs indicated in the figure to select for the next round of partitioning. In this simple case the answer is intuitively clear: subgraph 1 should be partitioned next, as it has the clearest division into two subgraphs.

The "next subgraph" decision rule may be made operational in the following way. First, the strength S of each subgraph i in the current decomposition must be calculated. There may be anywhere from 1 to $(n-1)$ such subgraphs. Then the subgraph with the smallest strength measure is selected as the next subgraph to be partitioned.

The logic behind this choice is simple: the lower the subgraph strength, the higher its propensity for being subdivided. Also, this rule is a reverse image of the rule for deciding which subset pair to merge next when clustering techniques are used in handling the decomposition problem (see Chapter 5, Section 5.3).

In the case of Figure 6.14, the strengths of the two subsets are 0.078 and 0.233 respectively. Thus the lowest strength rule would select subgraph 1 as the next subgraph to partition, which is certainly the correct choice in this case, as discussed earlier.

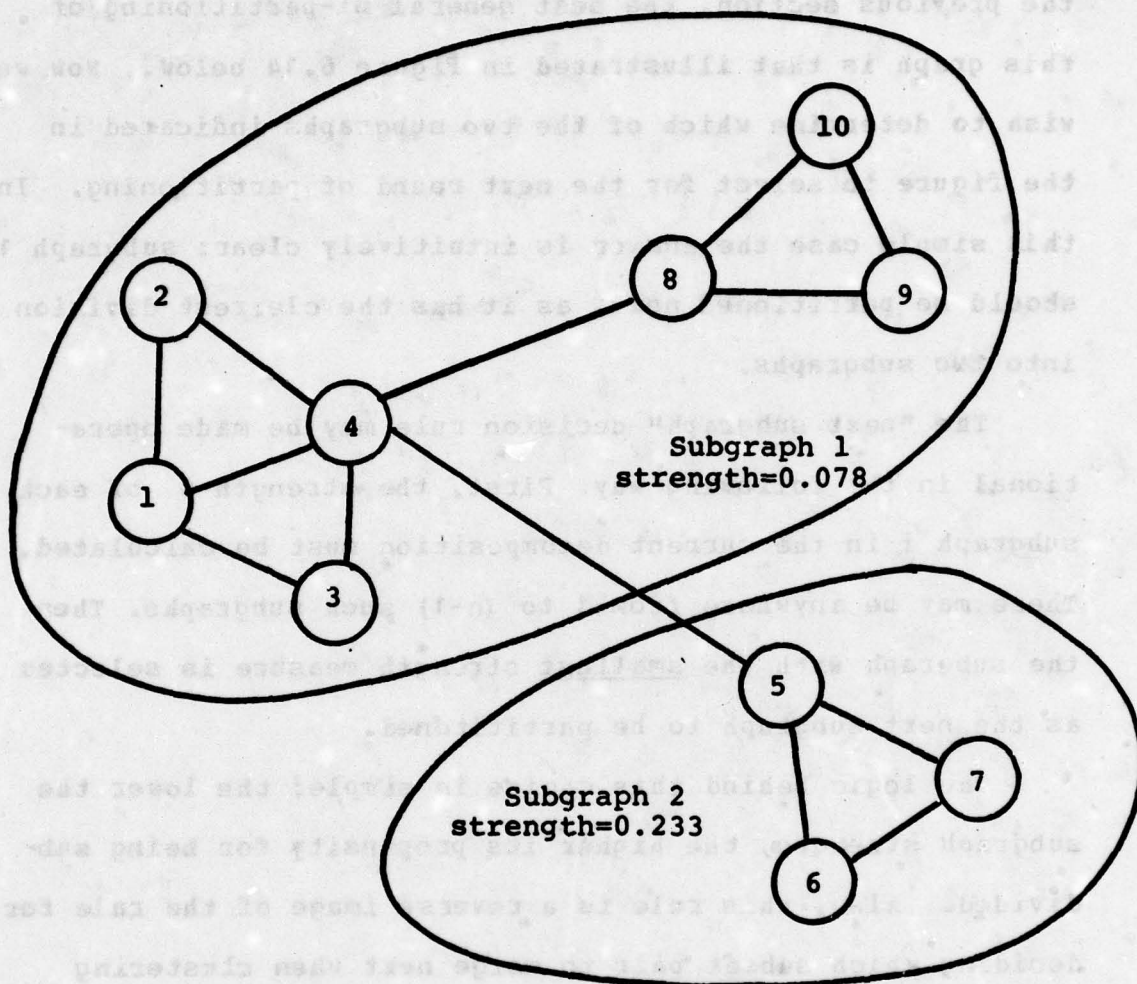


Figure 6.14

Partitions of the graph from Section 6.2 after one pass of the interchange algorithm, showing subgraph strengths.

6.6.2 A Stopping Rule for the Master Algorithm.

What stopping rule might be used to eventually halt the hierarchical partitioning procedure being implemented by the master algorithm? One obvious rule would be to continue partitioning until all current subgraphs are too small to be subdivided further - i.e., until $n_i < 2 \cdot n_1$, where n_i is the dimension of subgraph i and n_1 is the lower bound on subgraph size. Note that if $n_1 = 1$, the master algorithm would have to continue partitioning until all subgraphs consist of exactly one node.

It may be possible to devise a simple stopping rule that would halt the master algorithm well before this point is reached, however. Experience with partitioning a variety of graphs with the interchange method suggests that, generally speaking, the maximum value of M is attained after a fairly small number of "splits" have been made. In the above example, for example, only two splits were required to reach the optimum, whereas a total of nine splits were implemented before completely decomposing the original graph. Hence some simple rule such as stopping after k successive steps have yielded a lower value of M than the preceding step may prove useful and effective. A study of such stopping rules will be conducted in the future.

6.6.3 The Master Algorithm.

One possible implementation of a master algorithm for controlling the interchange partitioning method is given below.

1. Get data on graph structure, link weights; also get value of n_1 (subgraph lower bound).

2. Assign the entire graph as the current partition (CP);

$CP \leftarrow$ entire graph.

3. Calculate the measure M for CP;

$M_{opt} \leftarrow M; \quad CP_{opt} \leftarrow CP.$

4. Calculate the strength S_i for each subgraph i in CP. Locate the subgraph with the lowest value; assume this is subgraph ℓ .

(i.e., $S_\ell \leq S_i \quad \forall i$).

5. Call the Simplified Interchange Algorithm to partition subgraph ℓ .

6. Update CP: $CP \leftarrow$ (old CP with subgraph ℓ replaced by its sub-partitions as determined in step 5).

7. Calculate M for CP.

8. If $M > M_{opt}$ then do:

$M_{opt} \leftarrow M$

$CP_{opt} \leftarrow CP.$

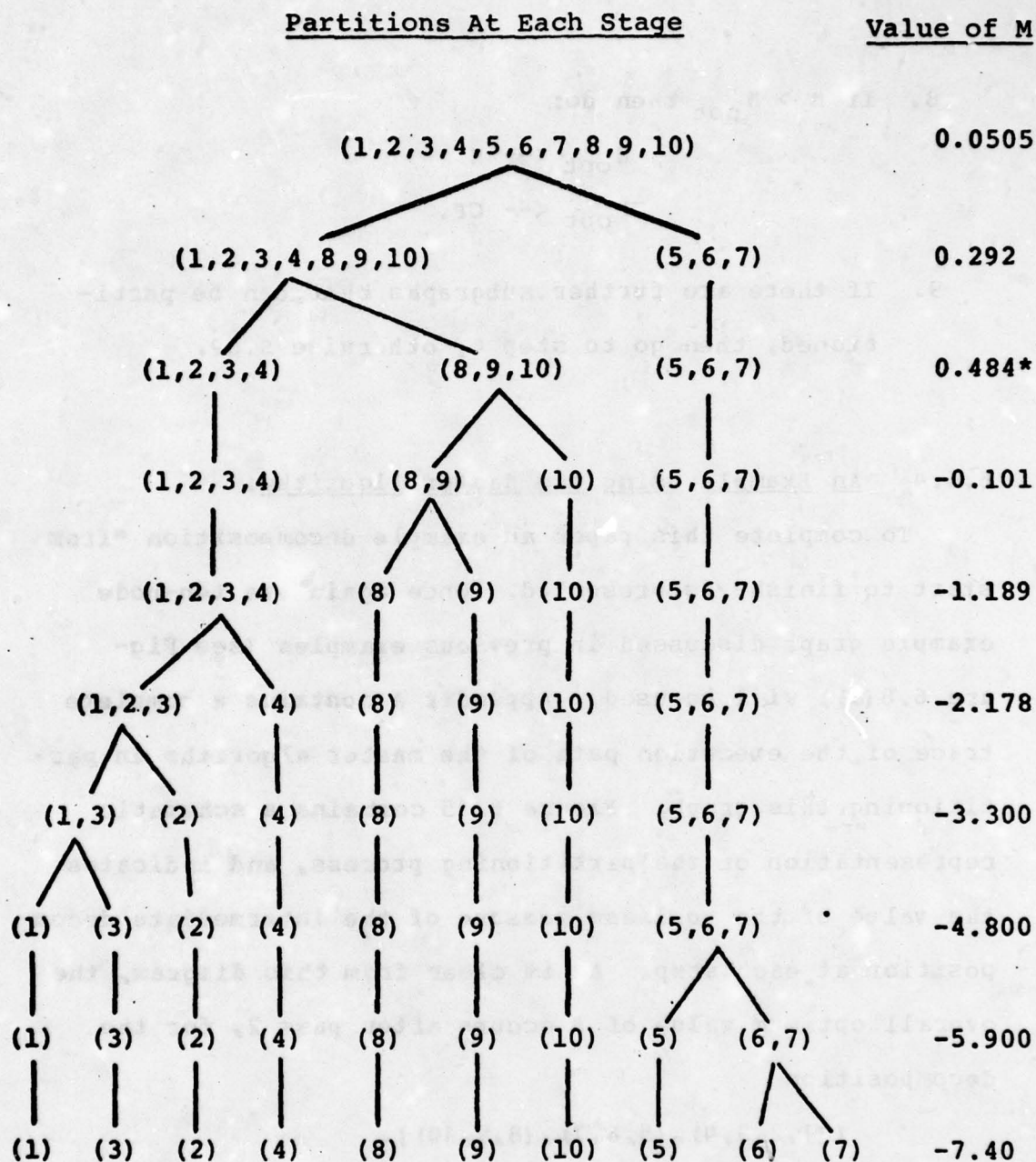
9. If there are further subgraphs that can be partitioned, then go to step 4; otherwise STOP.

6.6.4 An Example Using the Master Algorithm.

To complete this paper an example decomposition "from start to finish" is presented. Once again the ten-node example graph discussed in previous examples (see Figure 6.8(a)) will be used. Appendix A contains a complete trace of the execution path of the master algorithm in partitioning this graph. Figure 6.15 contains a schematic representation of the partitioning process, and indicates the value of the goodness measure of the intermediate decomposition at each step. As is clear from this diagram, the overall optimum value of M occurs after pass 2, for the decomposition

$\{(1,2,3,4), (5,6,7), (8,9,10)\}.$

The optimum decomposition measure is $M=0.484$.



*optimum measure

Figure 6.15

Decomposition "tree" showing complete decomposition
of graph from Section 6.2.3.

6.7 SUMMARY.

A new algorithm for performing top-down hierarchical partitioning upon a weighted graph has been defined and discussed. The algorithm functions by performing pairwise node interchanges and calculating the corresponding gain to the partition goodness measure M that results. The node pair exchanges that produce the maximum gain are retained, and those nodes are marked as ineligible for further interchanges, then the process repeated until no available node pairs remain. The sequence of node pair interchanges that leads to a maximum partial sum of the maximum gain values is then implemented. If the maximum partial sum is zero, then no interchanges are effected and no further improvement is possible.

The interchange algorithm is "driven" by the partition goodness measure M . In order to improve the efficiency of the algorithm certain approximations to the calculation of M were introduced. These approximations were verified by way of example, and were shown to lead to an equivalent set of interchanges in one example problem. Experience with the approximate gain technique has shown it to be essentially as effective, in terms of identifying good decompositions, as is the exact technique. The approximate technique is on the order of 100 times faster than the exact technique.

A comparative analysis in Section 6.5 shows that the interchange dominates all the hierarchical clustering techniques in 13 out of 14 cases studied. In the one case where interchange did not dominate, it did determine the second-best decomposition of the five techniques being studied.

A "master" algorithm appropriate for controlling and driving the node interchange algorithm was also introduced. A complete trace of the decomposition of a 10-node graph, using this master algorithm and the approximate interchange technique is included in Appendix F.

It is believed that the interchange algorithm is a useful contribution to the body of general graph partitioning techniques. In the present context (SDM), it is especially powerful, as it may be used as the core of a top-down partitioning search for the optimal graph decomposition. Since our experience with SDM has consistently shown that the optimal decomposition tends to reside fairly near the top of the decomposition tree (see Figure 6.15), a top-down search will generally reach the optimum fairly rapidly as compared with bottom-up clustering approaches used in this work previously.

* * * * *

In the next chapter, we turn from analytical work on SDM decomposition to a real-world application of the methodology. This application involves the architectural design

for a new budgeting and planning system for M.I.T. Chapter 7 contains detailed description of the system under design, as well as the process of applying SDM, and the lessons learned therein.

Chapter VII

THE USE OF THE SYSTEMATIC DESIGN METHODOLOGY IN THE DESIGN OF AN APPLICATION SOFTWARE SYSTEM.

7.1 INTRODUCTION - THE NEED FOR SDM EVALUATION.

The Systematic Design Methodology research to date has involved both methodology development and application studies. The applications addressed in earlier reports include designs for a database manager (Andreu 77(a)) and an operating system (Holden 78). In both cases, however, these studies were carried out by SDM researchers themselves - not by the "real" system designers. For this reason, they presented a somewhat biased result. For one thing, the individual performing the study was already very familiar with the methodology itself, its goals, and operational features. Thus there was little or no designer learning time involved (there was, however, learning time as regards the application being addressed). Furthermore, while these investigators did both report that using SDM seemed to provide both direct (an effective architecture) and ancillary (a better understanding of the system requirements) benefits, the credibility level of their assessments must be judged somewhat lower than would be those of a real system designer operating with a real design problem.

In order to determine how well SDM would perform in a real design context, and to learn how a practicing system architect would view and evaluate it, we undertook to locate an appropriate scenario within which to carry out such a study. Fourteen organizations were contacted by letter and then by telephone, and five indicated they (a) currently had an appropriate design problem under consideration, and (b) would be willing to spare the manpower necessary to carry out such an evaluation. One of these organizations was M.I.T.'s own Business Systems Development office (BSD). It was felt that BSD was the best choice for an initial outside application study for three different reasons. First, communication and transportation problems would clearly be nonexistent (all the other organizations were located in distant cities). Second, following an initial presentation of the concepts and objectives of the SDM, the BSD people concerned seemed genuinely interested and willing to expend some effort in a serious evaluation of the methodology. Finally, the system deemed most appropriate for the evaluation scenario was a fairly conventional, yet reasonably complex, data processing application system. As the earlier SDM applications had been concerned with systems software - a database management system and an operating system - this study promised to provide new insights as to SDM applicability to such application systems design.

This investigation, then, provides the first significant unbiased evaluation(17) of the usefulness and effectiveness of the methodology. Also, in return it provides the BSD system designers with an SDM-derived architecture upon which they may base the further detailed design and development of their target system.

The remainder of this chapter is organized as follows. In the next section, background information on the target system, a new M.I.T. computer-based budgeting system, is given. Section 7.3 contains a discussion of the process of applying the SDM to the Budget System architecture design, and includes certain observations made by the BSD designers, as well as lessons learned by the SDM researcher, in the course of working through the application. Section 7.4 describes the results of the graph decomposition calculations, and presents the system architecture that emerged from the SDM analysis. Implications of the suggested architecture are also discussed. Finally, the important lessons learned from this exercise are summarized in the Summary, Section 7.5. Appendices G through M include various exhibits pertaining to the analysis and decomposition exercise, including original and final sets of requirement statements, and the interdependency assessments.

(17) in the sense that the assessment data comes from real system designers, not the SDM researchers.

7.2 APPLICATION SYSTEM BACKGROUND - THE M.I.T. BUDGETING SYSTEM.

In this section we provide brief background information on the specific application system being addressed in the study. The focus is a computer-based system to support the M.I.T. budgeting process. This system will be referred to as the Budgeting System. A clear distinction must be made between the present budgeting system, which is also partially computer based, and the new system being designed. Both the present system, and considerations for the new one, will be discussed below.

Much of the information presented below was gleaned from two sources: a Sloan School of Management Master's Thesis written by M. Gutierrez and U. Schirmer which provides a detailed description of the present M.I.T. budgeting process, and supporting systems (Gutierrez and Schirmer 77); and, especially, a report written primarily by the chief designer of the new M.I.T. Budgeting System, H. von Letkemann (von Letkemann 78).

7.2.1 Current M.I.T. Budgeting Environment.

In this report the terms budget and budgeting are used in a broad context. They include financial planning and financial management, and therefore overlap with other elements, general planning at one end of the spectrum and spe-

cific accounting or reporting at the other. The terms include, but are not limited to, the existing Institute budget system, financial target setting procedures, forecasting of financial requirements, local departmental budgeting systems, and generation of various financial reports.

Budgeting functions at M.I.T. take many forms. In this report these functions are divided according to the three levels of management primarily concerned with them. The titles listed for these three levels are examples and are not meant to be all inclusive.

1. Top Management - concerned with Institute-wide planning and management. This group includes the President, Chancellor, Provost, Treasurer, certain Vice Presidents and the supporting Finance and Budget Offices.
2. Senior Management - concerned with planning for, and management of, specific major components of the Institute. This group includes the Deans, Vice Presidents, Department Heads, and the Directors of Laboratories, Centers, and programs.
3. Administrative Management - concerned with carrying out the plans and supporting operations of senior management. They include Administrative Officers, and certain Administrative Assistants.

At M.I.T. an overwhelming number of demands for funding compete for a finite amount of resources. The Institute has a fiscal 1979 operating budget of \$336 million. Of this amount approximately \$200 million is direct expense for sponsored research, \$55 million for instruction and unsponsored research, and the balance for support services and

auxiliary activities. There are about 130 budgeting entities consisting of schools, academic departments, interdepartmental laboratories and centers, senior officers, support departments and special activities. The active accounts number about 10,000. Resources must be allocated among those programs in a manner consistent with the academic and societal goals of the Institute.

The Institute faces substantial fiscal pressures and constraints, both internal and external. It has considerable fixed expenses, including an extensive physical plant and a 60% tenured faculty. Recent shifts in enrolment patterns have strained the capacities of some departments and led to underutilization of others. Externally, the impact of inflation has been substantial. The cost of materials and services has gone up every year, and salaries and wages have been increased in an attempt to keep pace with the increased cost of living. Inflation has also aggravated a second key problem, the economic slowdown that the United States has experienced for the last several years. Although there are some indications of recovery, many sources of gifts and research sponsoring agencies, including government agencies, corporations, foundations and individuals, are still holding back because of their own economic problems. Additionally, problems such as the ever worsening energy crisis, and additional government regulations and requirements continue to burden the Institute's limited resources.

M.I.T.'s responses to these economic problems have taken several forms. Budget reductions have been necessitated in every area. For the most part these budget adjustments have been absorbed without detriment to the services provided. However, there is the general feeling that the easy cuts have been made and that future reductions will be more painful.

Major efforts are under way to develop new sources of recurring income and gifts to be used in operations. The Leadership Campaign, and the expansion of the Industrial Liaison Program, efforts by faculty to secure more sponsored research support, all are directed toward this goal. Various other Institute programs, including the new Facilities Management System to optimize building energy use, and increased undergraduate enrolment, have also been introduced to achieve further economies or additional income.

Some of the financial challenges which M.I.T. faces will change with time and others will remain the same. New problems will arise and old ones will be solved, but it is clear that the Institute must use its resources wisely and efficiently if it is to continue to meet its goals of excellence in education and contribution to our society. With these goals in mind and the knowledge that resources are limited, it is essential for M.I.T. to have a good system for budgeting and financial management.

7.2.2 The Existing Budgeting System.

The budget system now used at M.I.T. grew as a result of responses to specific requirements. As a need was recognized a new component of procedure came into existence. The system is soundly based on the M.I.T. account structure and includes some analysis functions. These characteristics make it a valuable guide for any new system. However, it is still a loosely-connected mixture of manual procedures and computer operations. The system has not been developed sufficiently to take advantage of the available data already in the budget files and in those of the Accounts Reporting System (ARS). Other important data, particularly historical data, is not even in these files and must be developed manually from various sources. The functions of the existing budget system are hampered by the lack of an integrated base of consistent information. This has kept it from being the important management tool it could be. Some of the limitations of the existing procedures are discussed from the viewpoints of the various levels of management.

7.2.2.1 Top Management.

The reports used or issued by Top Management at M.I.T. are predominantly manually produced. They are frequently prepared in response to changing requirements. Often the pertinent data does not exist in a computer-based file, or

if it does the format or content may be inconsistent with other files. Even periodic reports, such as the Treasurer's Report, the Operating Budget, the calculation of research overhead, the M.I.T. Operating Plan (MITOP), and the Dynamic Model(18) are produced either entirely or partially by hand.

The manual preparation of a report does not necessarily detract from its value or content, but often this preparation requires extended periods of scarce managerial time. Production of reports either entirely or substantially by computer would use Institute resources more efficiently.

Cumbersome manual methods of handling information have a real impact on what information is used and what is done with it. For example, the Dynamic Model forecasts Institute financial scenarios several years into the future by projecting current data and assumed trends. Because of the time and difficulty involved in changing the assumptions and running additional iterations, only a limited number of combinations are reviewed. If the model could be changed more easily, more combinations of variables, and their relative impacts, could be assessed and it could be run more often. Then managers could spend their time more effectively in steering controllable elements and monitoring important external factors.

(18) For additional detail on these and other components of the current budgeting system, refer to (Gutierrez and Schirmer 77).

There is no system for looking ahead several years by collecting, evaluating and summarizing the planned activities and expense projections of the senior managers of the Institute on a regular basis. Even when setting budget targets with the Chancellor there often has been little attention paid to the years beyond the period being budgeted. Although many senior managers do their own longer range planning, these plans and projections are never brought together to show the aggregate of the estimates and their effect on where M.I.T. will be three to five years hence.

Fund accounts are frequently managed in a less than optimal fashion. For instance, an unrestricted gift may be received and then designated for a specific use by the Institute. The fund is then accounted for according to that designation. In time that designation may begin to lose priority. With no easily accessible record to show that it was originally an unrestricted gift, there is no way to be sure that the gift is being used to the Institute's best benefit.

7.2.2.2 Senior Management.

As with top management, senior management must rely heavily on the current and previous year's figures when developing their future budget plans. Although certain items in their budgets will be adjusted by the Budget Office

to reflect salary and tuition increases and other changes, it is sometimes difficult to know what resources will be required for the coming year, particularly if any changes in activities are planned. The President, Chancellor and Provost give general guidance, but they depend heavily on the judgement of the deans regarding new subjects, trends in student demand, and research undertaken. The absence of uniform planning and budgeting presentations allows for a significant amount of subjective judgement to be exercised in the establishment of the budget targets.

In the cases where a request of the senior manager for a budget increase is accepted, it is likely that the request has been supported by a detailed and well structured projection explaining the requirement. Although no detailed justification nor any plan beyond the next fiscal year is normally required, it is often those managers who document their needs and provide the most meaningful presentations who get the most consideration for additional funding. However, the current budget system provides almost no effective support to those managers who are motivated to develop such thorough documentation.

7.2.2.3 Administrative Management.

Administrative management is the group closest to the day-to-day financial management of the Institute. As a group, it has the greatest need for current and detailed information about individual accounts. This function is supported by the Institute with periodic reports such as those listed below.

The Accounts Reporting System (ARS) provides them with:

Detailed Transaction Report

Monthly Statement

Information Summary

Volume Report

Analysis of Expired and/or Overrun Accounts

The Budget Office provides them with:

Budget Proposal Forms

Budget Authorization (green sheet)

Budget versus Actual Analyses

The Payroll department produces:

Salary and Wage Expenses by Individual

Consolidated Salary Expense Analysis

The ARS reports contain essential accounting data, including information regarding monthly charges, fiscal year and cumulative figures, and authorized budgets. Commonly mentioned shortcomings of ARS reports are that the data is not timely and that the commitment figures are not always meaningful.

These problems are inherent in the design of the current accounting and purchasing systems.

Budget vs. Actual reports produced by the Budget Office are the only real analyses that the Institute provides the administrative managers. These reports compare fiscal year-to-date expenditures with Budget Office projections based on standard expenditure patterns. While the projections are generally sensible and realistic, not all administrators find them useful. Under the current system however, these reports are probably the best that could be produced on an Institute-wide basis.

In many instances, individual departments have developed their own tailored systems to monitor actual-versus-budgeted expenditures or provide other services deemed important by that department. The scope and sophistication of these "local" systems varies widely. However, their existence indicates the existence of a multitude of reporting and monitoring needs not now met by the current budgeting and accounting systems. They also represent a rich source of ideas for potential features of a new budgeting system.

7.2.3 General Requirements for a New Budgeting System.

In this section, an overview of various user requirements for a new budgeting system is presented. These requirement issues are derived from many sources, including interviews with management personnel across the Institute, other interviews and questionnaire survey results from a study of the planning and budgeting practices of eleven other colleges and universities (Hudock 77), and analysis of the current M.I.T. Budgeting System operational capabilities.

The new budget system will build on many strengths of the existing Budget system and the systems developed by several of the administrators. It will automate many of the manual procedures and extend the present system's capabilities by increasing the data available to both the Budget Office and the departmental users. Capabilities could be expanded by sharing the data bases of other systems and by making budget data available to users in other areas. Some additional input would allow improved support for a broad range of financial management applications and additional reporting capabilities. These features are discussed in this section in the context of the management level primarily involved.

7.2.3.1 Preliminary Technical Issues.

The present budgeting system is batch-oriented and heavily involves magnetic tapes for data storage. The new system would provide for considerable on-line function, as well as batch, and would rely much more on disk storage media. Tapes may still be used for disk backup, and possibly for transferring data between other older systems.

The new system will be developed and operated on one of the Institute's I.B.M. System/370 computers. Storage for the proposed database will require on the order of one full disk pack (3330-1 type). The system will be able to interface with different terminal types so as not to constrain the system users. Any terminal which normally communicates with the 370/168 should be acceptable. A new printer, the IBM 3800, is desirable for the new system. It would allow more data to be shown on a report, could produce the reports in less time, and would print them on 8 1/2 inch by 11 inch paper, which is easier to handle and store. A sample print-out from the IBM 3800 printer is included in Appendix G.

The new system will be designed to operate in conjunction with a database management system. The database management system (DBMS) will support storage of detailed information and allow simple access and updating through batch or interactive processing. The DBMS will support standard and non-standard reports and inquiries, and func-

tion independently of the programs and systems using it. It is planned that the database management system will interact with many systems, increasing its usefulness beyond just the budgetary function.

7.2.3.2 Support for Top Management.

These functions are categorized in three areas: special information requirements, standard reports, and planning.

Special information support for top management basically involves supporting the need for "one-time" reports or queries. Although it is not feasible to anticipate every request for such information, the Budget System must carry a wide range of data that can be easily accessed, organized and presented as required. The details of this function are somewhat dependent on the capabilities of the database management system used. It is anticipated that the data would include at least the Chart of Accounts and detailed monthly budget and actual figures for each object code for each account. Certain data for past fiscal years would also be included. For fund accounts there would also be historical information that could facilitate their management. For example, fund data should include the donor's original designation for the gift and its related income as well as how it is currently being used, thereby making more effective fund management possible.

The new system would continue to produce most of the current standard reports, including (but not limited to):

The Printed Budget

Certain portions of the Treasurer's Report

Indirect Cost Recovery Percentage

MITOP

Dynamic Model

Periodic Summary of Operations

Modeling and analysis capabilities must be provided to explore historical data and to project observed trends and assumptions. This would probably require a new program to replace the Dynamic Model, which would automatically interrelate the various assumptions. This would facilitate re-running the model so as to check out assumptions or do sensitivity analyses on individual factors. This modeling and analysis system could be developed in-house or a commercially available package might be used.

If M.I.T. is to take full advantage of the new system's modeling and forecasting capability there must also be input concerning the plans made by top and senior management. The budget system would provide the support for collecting, storing and providing access to such data. The most important contribution to a forecasting system would be senior managers submitting their plans and projected expenses related to those plans. These should be for two specific

periods; for example, a "short range" one year plan and a "long range" three year plan. The plans should be in a reasonably uniform format and should be correlated with proposed budget targets as well as the senior managers' area summaries in the Report of the President and Chancellor. The Budget and Fiscal Planning Office would then collect these plans and projections and enter them into a planning database to support modeling and forecasting.

7.2.3.3 Support for Senior Management.

The Budget System would provide senior managers with standard periodic reports, special reports and access to the database that would allow them to make their own inquiries and analyses. These reports would contain data extracted from the Budget System database or any other file which is normally accessible.

Of the standard periodic reports currently produced by the Budget Office only the Budget Authorizations issued prior to the beginning of the fiscal year would remain the same. In the new system the subsequent budget authorizations and changes would be included in a Monthly Analysis report. The Monthly Analysis would also replace the Budget vs. Actual Report. This report would be a summary of the analyses produced for administrative management.

Special reports for senior management would be available on request. They would include variations of the Monthly Analysis report and other widely-used reports. It is anticipated that they would be requested via a terminal and printed either at the terminal or on a high speed printer at the data center.

Customized reports could be obtained by use of an easy-to-use Report Writer language to access the database. Senior managers would be able to access their data, perform various kinds of calculations, and display the results in a variety of formats.

The Budget database would also be available for special inquiries or analyses originated by senior managers. There would be support for batch processing as well as a pre-programmed "menu" for terminals which would allow easy access to the database for the most common types of inquiries. More complex analyses could be obtained by using an easily-learned database inquiry language.

Protection of data against unauthorized access would most probably be done by a system of passwords. Within a department there might be several levels of security depending upon the sensitivity of the data and the "need to know." Furthermore, even when data elements would be accessible to authorized users, most of them would be on a "read-only" basis. To maintain database integrity, only the data

"owner" - such as ARS, Payroll, or Budget Office - would be allowed to add or change most data.

As for top management, there is a need for senior managers to submit their future plans and projected expenses. Not only will this aid top management in modeling and forecasting, but it will also assist senior managers in presenting a concise, meaningful and convincing proposal for their financial support.

7.2.3.4 Support for Administrative Management.

Just as the administrative managers have the most intimate and continuing contact with budget and accounting functions, they would experience the greatest impact from the new budget system. The system would make considerably more data available and would provide facilities to access it. It would also demand more of them, in that to effectively use the system they must provide, and revise as necessary, month-by-month projections of expenditures and income for each account and object code. The system would make this as easy as possible to do. Each object code would have a standard or "default" projections formula which the administrators could either accept or replace with their own. Projection changes within an object code would be made directly by the Administrative Officer and reviewed by the Budget Office. Other changes to the budget data would be submitted to

the Budget Office, which would be responsible, as it is today, for review prior to updating the database.

Program Budgeting can be an effective tool in relating plans or goals of the Institute and senior management to the financial resources available. It is a method by which budgets are established along program or activity lines. Although some administrative managers use Program Budgeting, others budget and monitor solely on a line-item basis. The new budget system should encourage the use of Program Budgeting. This budgetary method would be far more useful in monitoring expenditures than the traditional line-item budget. In addition, program budgeting would be a significant aid in estimating requirements and in preparing for the target-setting discussions between senior managers and the Chancellor.

The Budget System should provide manual and on-line options for the preparation and submission of budget proposals. Duplication of effort, and time to prepare proposals, would be minimized by using the computer to do the calculations and make projections and modifications within the budget target amount.

The new system would supply all the periodic information currently contained in the Budget Authorizations (after the start of the fiscal year), Monthly Statements, Information Summaries, and the Budget vs. Actual reports. This

would be done with a single Monthly Analysis report which would show current month, fiscal year to date, budget and other data in a format which would compare actual and planned account activity.

In addition to replacing these Institute reports, the system could eliminate the requirement for some of the departmentally-produced reports. If a department still wished to have its own special formats, they could do so by extracting their information from the database using the Report Writer feature.

The Budget System would produce optional reports on request. These would include standard reports for non-standard periods, such as contract year, or reports which would be widely, but not universally used.

The system would support the additional needs of administrators for inquiries into the database or for special analyses. Access to the data would be read-only, and, subject to data security restriction, would use the same facilities available to senior management.

7.2.3.5 Support for the Fiscal Planning and Budgeting Office.

The new budget system would cause some significant changes in the activities of the Fiscal Planning and Budget Office. In addition to most of its current res-

possibilities, there would be the establishment and maintenance of a database for the long range projections of the senior management. The dollar amounts and other volume figures should provide the Budget Office with the base for a good forecasting system.

Processing of budget proposals would be simplified by the use of computers and terminals, greatly reducing the routine manual functions. Proposals could be accepted either on paper or via department terminals. In either case, the computer would edit them for internal consistency and check or generate necessary totals. The computer would determine if the proposals were within the authorized target amounts and also check for open accounts without proposals. Any discrepancies would be followed up by the Budget Office.

The current procedures of written requests and approvals for nonrecurring equipment, carryforward amounts, sabbatical leaves and other special expenses would remain unchanged. The approval actions would enter the budget system as authorized budget changes.

The existing Fund Draft procedure would remain in effect, except the input log sheet would be replaced by a similar record entered via a terminal or batch input by the managing department and checked by the Budget Office.

The budget proposals accepted and approved by the Budget Office would continue to be adjusted for "dollar budgeting" via the computer, and Budget Authorizations ("green sheets") would be printed and distributed as at present. Once the fiscal year begins, any subsequent adjustments would appear on the new Monthly Analysis report produced from the Budget database. A note explaining the change would also be shown.

It is anticipated that the Budget database would have month-by-month figures for:

Proposed budget, next fiscal year

Authorized budget, this fiscal year

Actual expense, this fiscal year

Authorized budget, last fiscal year

Actual expense, last fiscal year

Summarized data would be included for prior years. The database would also carry, or be able to access, the data from the Chart of Accounts, account makeup and non-standard support, and additional data as required for fund and research accounts. Non-standard financial agreements between top and senior management, and other nonrecurring transactions, would be catalogued in a special Budget Office file. The database organization must allow the addition of data elements that are not currently required so that the database can grow and change with the needs of the Institute.

7.3 SDM ANALYSIS OF THE NEW BUDGETING SYSTEM.

In this section we describe briefly the steps that were taken in conducting the SDM analysis of the M.I.T. Budgeting System, and the methodological lessons learned. The key documents developed or referenced during this activity are contained in appendices.

As mentioned earlier, the SDM researchers' "intervention" in the Budgeting System design activities commenced with a presentation on the nature and purpose of the methodology, attended by the M.I.T. BSD staff. Following the presentation, it was agreed by the researchers and the BSD staff that the Budget System was probably the most appropriate system to use as an SDM test scenario. The main reason for this was that the system's development was at the right stage - i.e., most user requirements had been determined and documented, although detailed design activity had not yet commenced. Also, the system was perceived to be about the right size and scope for an effective SDM study: large enough to present considerable complexity to the designers, but not so large as to overwhelm the SDM researchers.

7.3.1 Requirements Preparation.

The first step in the study was to prepare a set of SDM-oriented requirement statements for the new system. Following initial discussions with the Budget System design-

ners, it was decided that the designers would prepare an initial requirements list, which would later be modified, if necessary. This initial list of statements was prepared by the two key Budget System designers, H. von Letkemann and R. Shaw, and is reproduced in entirety in Appendix H. These requirements statements were developed largely out of existing prose documentation of the needs of the various Budget System user groups, similar to the description given in Section 7.2.3

This initial set of requirement statements proved somewhat inappropriate for SDM use for various reasons. The most important difficulty concerned the manner in which many of the statements had been constructed by the designers. As may be seen in Appendix H, many statements consisted of a very general "leader" statement, followed by a series of sub-statements. For instance, original statement 19 was

19. Support Special reports for budget-related activities.

- a) Standard reports at non-standard times
- b) Standard reports for non-standard periods
 - i) Contract period
 - ii) Sponsor's fiscal year
- c) Standard data in non-standard formats
- d) Report writer language for fully customized reports. This language must be easily learned and used.

Also, a number of the statements included reference to implementation mechanisms, something to be avoided at this stage in system design. As an example, original statement 18 read

18. On Personnel Action Form add a box to indicate whether person hired is a replacement or an addition.

It is clear that as stated, this requirement specifies a procedural technique rather than a function to be provided.

Finally, the various statements exhibited wide variations in their abstraction level. Statement 1, for instance, originally read

1. Automate as many manual procedures as feasible to save time and effort.

There is a rather substantial difference in abstraction level between statement 1 and statement 18 (above). In fact, statement 1 was later removed, as it was felt to be so all-encompassing as to be design-irrelevant.

Occasionally, the designers' original set of requirement statements included requirements for issues to be studied further, as opposed to the functions of the target system. For example, original statement 37 read

37. Determine the desirability and feasibility of encumbering salary and wage budgeted amounts.

Statements such as these were judged to be "study tasks," and were not included as system functional requirements in the final set of statements.

A two-stage approach was followed for re-writing the set of functional requirements to work them into a form more suitable for additional SDM analysis. In the first stage, the SDM researcher re-drafted all the statements following the general guidelines discussed in Chapter 3. The templates concept was followed in framing individual statements, and proved quite effective in helping to meet the guidelines.

In the second stage, the designers examined the re-drafted statements to make additions, corrections, and modifications. This took place over the course of two meetings, of about two hours each. One interesting phenomenon occurred at this point. In many instances, the designers possessed specific, often implementation-oriented, information that bore upon certain requirements. They felt that it was important that such information be included within the requirement statements themselves. However, in many cases it was precisely this kind of detailed, implementation-oriented information that the requirements had been redrafted to avoid.

One of the underlying principles of SDM analysis is that requirement statements should specify functions only, not procedural issues. Including procedural information ("implementation issues") in the requirements tends to unnecessarily constrain later design options at the start, perhaps resulting in potentially superior alternatives never being considered. However, in this case the designers were effectively saying that various good procedural ideas had occurred to them, and they would "like to see them reflected in the requirement statements." Certain other factors played a role in the matter as well: wanting to include reference to a "pet idea" of particular users; wanting to include references to specific techniques or devices for which it was felt that higher authorities might require some "selling."

An effective solution to this problem was to add a Comment section to many of the requirement statements. This feature allowed the designers to include additional information, deemed not appropriate or relevant for the basic requirement statement, but which they desired to have formally stated along with the basic statements. Examples are contained in Appendix I.

The two meetings mentioned above led to a reasonable set of functional requirement statements for use in further SDM analysis. However, this was by no means the final ver-

sion of the statements. In the meetings to follow, numerous additional modifications to both statement form and content were made. Certain new statements were added in light of improved understanding that occurred as a result of these discussions. Similarly, some other statements were deleted or merged together, and minor or major wording alterations were made to many. The final version of the Budgeting System Functional Requirement Statements is given in Appendix I.

Another mechanism found useful in the development of the requirement statements involved the use of the Waterloo Script text formatting system ("WSCRIPT") which runs on M.I.T.'s IBM System/370 computer. This powerful formatter allows the user to write command macros. One such macro was used to provide automatic numbering control on the requirement statements. Through this means, statements could be added, deleted, or their sequence altered, without requiring extensive and time-consuming statement renumbering.

7.3.2 Interdependency Analysis.

Once the requirement statements had been expressed in a form appropriate to SDM, work began on determining the existence and strength of the various requirement interdependencies. This work was carried out in a series of six joint meetings, each lasting about two hours.

A simple form was designed for carrying out the interdependency analysis, a copy of which is included here in Appendix J. The approach followed was straightforward. Beginning with requirement pair (1,2), each individual considered whether or not a significant implementation interdependency existed between the two requirements. This assessment was carried out by considering "conceptual models" of the implementation of each requirement in the pair, then determining in the context of these models whether or not there would arise any concordant or discordant interaction between them. These notions of mental implementation models, and concordant and discordant interdependencies have been described in depth in (Andreu 77(b)). The basic idea is as follows:

1. First, one thinks about how the first requirement would most likely be implemented. This generally requires thinking through some detailed design, procedural-type issues.
2. With that "mental model" in mind, the same thing is done as regards the second requirement.
3. Then the two mental models of implementation are jointly compared to determine whether
 - a) one scheme makes it easier to implement the other (condordance);

b) one scheme makes it more difficult to implement the other (discordance);

c) there is no appreciable overlap, or interaction, in the above sense, between the two.

The result of this comparison suggests the existence or non-existence of an interdependency between the requirements under consideration.

4. Finally, the strength of each interdependency was assessed. Strength ratings were chosen from a set of three alternatives:

S - strong

A - average

W - weak.

While interpolation and extrapolation of these categories are possible, these three alternatives were found to be satisfactory for this project. The interdependency strength assessment was made judgmentally, based on the perceived amount of "overlap," or interaction, between the mental models being contrasted.

In practice, the different individuals involved in the assessment activity nearly always agreed on a common strength value for a given interdependency. Intuitively, then, these assessments should be judged to be reasonably consistent between different designers.

Interdependency analysis proved to be somewhat more difficult for the designers than expected. The main reason for this seemed to be the difficulty in constantly keeping in mind precisely what interdependency assessment was supposed to be. Specifically, there was a noted tendency for the focus to shift from issues about how two particular requirements might be related at the implementation level, to whether or not they were logically related. An example of this phenomenon should make it clearer. Requirements 56 and 57 are, respectively

56. Budget proposals can be prepared manually.

57. Budget proposals can be prepared on-line.

Now, on first glance it might appear that since both requirements pertain to budget proposal preparation, they must have an interdependency, probably a strong one. This would be an instance of what was termed "logical relationship," above. In practice, this kind of logical relationship is easier to identify than is the implementation-level interdependency, consequently they often "jumped out" at the designers during the interdependency analysis activity, tending to further obscure the search for true implementation interdependencies. The only solution to this problem

was for the SDM researcher to continually ask the designers whether a given interdependency they had determined to exist was in fact a result of implementation overlap, or something else. If the answer was "something else" (e.g., in the above example, the source of the initial interdependency assessment was "both requirements concern budget proposal preparation") then we had to think more carefully about the requirements and our mental models of their implementation within the target system. In the above example, this rethinking did in fact lead to an implementation interdependency, judged to be weak in importance. The underlying argument concerned a key implementation model, the concept of a suspense or holding file for budget proposals, that was seen as leading to a concordant interdependency between the two requirements.

7.3.3 Some Lessons Learned.

The interdependency analysis activity, as mentioned above, consumed approximately eleven hours of meeting time. This is a not inconsiderable load. However, in this case at least, the meetings were judged to be profitable exercises in a sense independent of any potential benefits that may emerge from the SDM-produced architecture per se. Specifically, some important issues regarding the Budget System were raised, discussed, and cleared up or at least better

understood as a result of the careful, repeated study being given to each requirement. This effect is raised and discussed, and modelled as an important SDM benefit in the next chapter.

The most general side benefit gained from the SDM analysis exercise concerned a heightened awareness and understanding of all the "pieces" of the new Budget System, and now they fit together. The designers indicated that working through the SDM activities, especially the interdependency analysis step, served both an integrating and differentiating function. Developing implementation-free requirement statements tended to force them to "stand back," to abstract from many of the specific implementation-oriented details with which they were generally concerned, thus helping them to develop a better grasp of the "big picture." An example of this concerns original requirement

- 5c) Provide checks to ensure that each person
is not budgeted more than 100% E.F.T.

Discussion of this requirement led to the broader recognition that what was really desired was a general set of editing and checking capabilities, not limited to this one particular aspect. Hence the more general requirement,

- 58. Budgeted proposals will be automatically

checked and edited to the extent possible.

emerged. Nonetheless, the problem with EFT (effective full-time) budgeting of certain staff occasionally exceeding 100% was felt to be an important instance of the general requirement, so was included in the final set of requirements as a comment.

Another class of lessons learned concerns new ideas that occurred to the designers as a result of working through the SDM analysis. A good example of this was related by the chief designer during one of the meetings. It concerned his observation of the parallels between the research proposal tracking and budget proposal tracking tasks. In the past these activities were viewed and treated separately. However, through having to think carefully about the relationships among the requirements in the course of the interdependency analysis, he had come to recognize many procedural commonalities between the two general activities. This, he pointed out, suggested new, potentially better ways of performing the former task based on ideas that had been developed for performing the latter. At the present time the procedures for performing the two tasks are quite different. Essentially, the need to develop a mental implementation model for the research proposal tracking requirement led the designer to consider a similar, better understood model for the budget proposal tracking

requirement, which in turn led to the idea of implementing both requirements in a common fashion. This may be thought of as a kind of inverted interdependency analysis: rather than deriving the interdependency from the conceptual implementation models, one implemented model was derived from the second model and the perceived potential interdependency. The normal and inverted patterns of interdependency analysis are illustrated in Figure 7.1(a) and (b).

A third category of benefit reported by the designers was that working with the SDM concepts gave them some useful new ways of thinking about system design in general (not restricted to this specific system). The most frequently cited case concerned the central SDM concept of separating functional issues in the requirement statements, and implementation issues in the interdependency assessments. The designers reported that they found this a most useful way of organizing their thoughts in addressing system design problems, and in fact found themselves using the concepts when discussing design issues with other parties. They reported conversations with the Director of Business Systems Development (their boss) in which the SDM conceptual framework was used to help clarify certain design issues being discussed.

Another category of "lesson" that ought to be mentioned concerns the importance of what we will call "political" issues in the system preliminary design process. As with

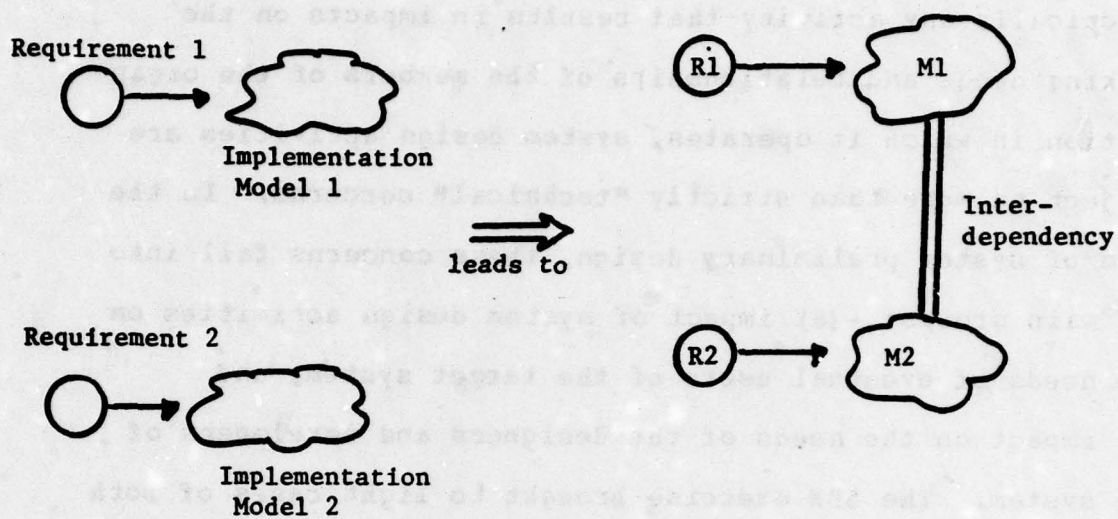


Figure 7.1(a)

Normal pattern for Interdependency Assessment

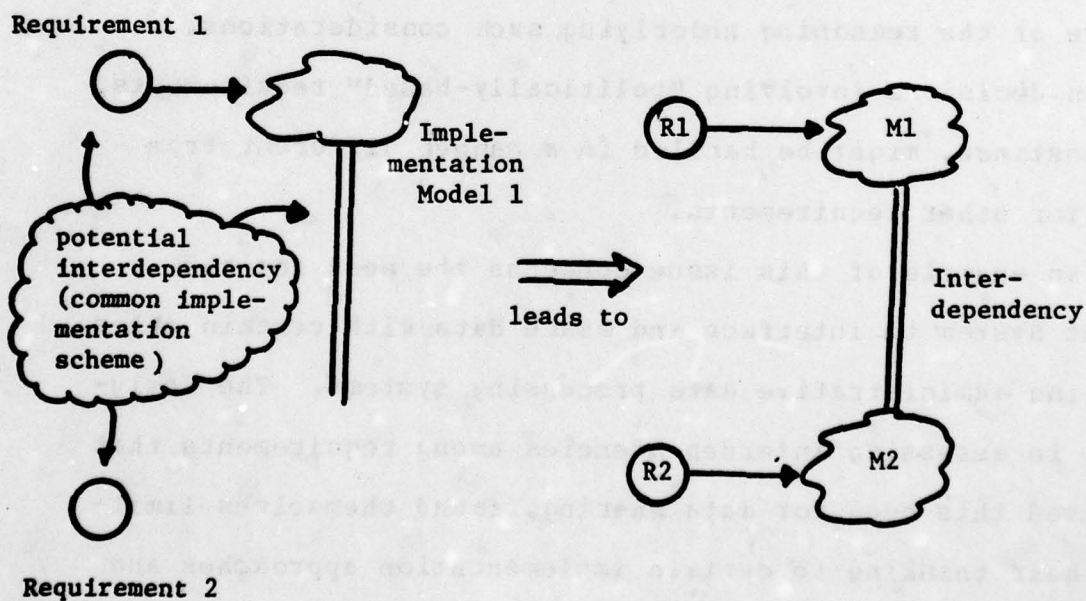


Figure 7.1(b)

Inverted pattern for Interdependency Assessment

practically any activity that results in impacts on the working needs and relationships of the members of the organization in which it operates, system design activities are subject to more than strictly "technical" concerns. In the case of system preliminary design, these concerns fall into two main groups: (a) impact of system design activities on the needs of eventual users of the target system, and (b) impact on the needs of the designers and developers of the system. The SDM exercise brought to light cases of both types. This was found useful by the designers, although not because they believed that these kinds of issues ought not enter the design process at all. In most cases the designers were not really in a position to make such a judgment. Rather, it was seen to be beneficial simply to recognize the nature of the reasoning underlying such considerations. Design decisions involving "politically-based" requirements, for instance, might be handled in a manner different from that for other requirements.

An example of this issue concerns the need for the Budget System to interface and share data with certain other existing administrative data processing systems. The designers, in assessing interdependencies among requirements that involved this need for data sharing, found themselves limiting their thinking to certain implementation approaches and ruling out other, potentially good approaches that were seen

AD-A074 911

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC METHODOLOGY FOR DESIGNING THE ARCHITECTURE OF COMP--ETC(U)
SEP 79 S L HUFF, S E MADNICK
CISR-P010-7906-12

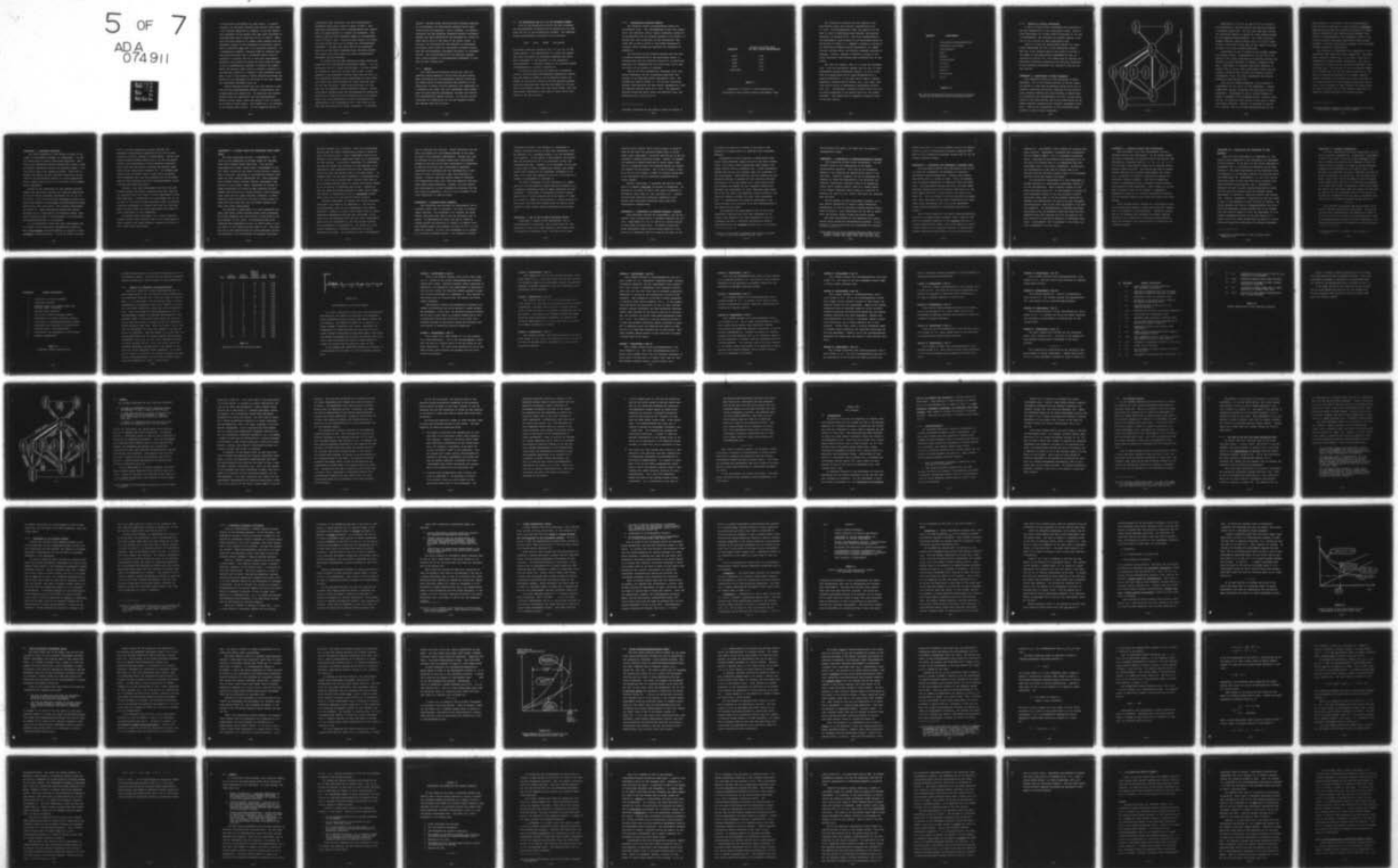
N00039-78-G-0160

NL

UNCLASSIFIED

5 OF 7

ADA
074911





as "politically infeasible" for some reason. In another instance, the designers suggested that certain items ought to be included (generally as comments) in the SDM requirement statements on the grounds that some other individual or organizational entity "would want to see it there." There were also some instances of comment items stemming from the designers' desires to give expression to particular techniques or approaches they felt to be especially important. As a hypothetical example, a designer might be convinced (perhaps quite correctly) that a particular device would be necessary to properly meet one or more user requirements. Therefore, even though the choice of device could be argued to be an "implementation approach" to meeting the requirements, the designer might choose to include a reference to the particular device as a comment on the requirement statement, so as to help develop a mental association between the device and the requirement in the minds of the users reading the requirement statements later on.

Andreu expressed concern over the time required to execute the SDM interdependency analysis on requirements sets of nontrivial size (Andreu 78). He countered this concern, however, with the observation that the interdependency matrix is quite sparse, hence the problem is not as serious as it might at first appear. This turned out to be accurate in the present case as well. For the Budgeting System, 77

requirements were determined, and 289 interdependency assessments made over a course of about 12 hours. This represents approximately four interdependencies per requirement, and approximately 2.4 minutes per assessment. Note however, that the total potential number of interdependencies is $77 \times 76 / 2 = 2926$. Using the total figure, the assessment rate turned out to be 15 seconds per interdependency assessment. The fact that about 90% of the requirement pairs are of the "easy" assessment type, and hence require very little study time, makes the entire interdependency assessment activity feasible.

In carrying out his DBMS application study, Andreu performed all the interdependency assessment himself - i.e., he played the role of a single DBMS designer. He later pointed out (Andreu 78, page 232) the fact that he felt a group approach to the assessment activity might work out well, in that individual designers need reinforcement of their thinking process from other designers to insure them that they are not "way off base." This in fact did seem to be the case with the Budgeting System assessment exercise. Having three people thinking about the interdependencies definitely resulted in a clearer and more consistent set of interdependency, and in the propagation of ideas, modification and improvements to the requirements, etc. that would not all have been generated by any single individual. An effective

balance - between target system-relevant knowledge possessed by the designers, and SDM-oriented concepts better understood by the SDM researcher - was in evidence. On numerous occasions, the SDM researcher suggested possible interdependencies that were discounted by the designers as a result of their better grasp of the needs of the target system. In contrast, the SDM researcher was effective in maintaining the correct focus during the requirement statement development and interdependency assessment activities, as discussed earlier. The materialization of this symbiosis suggests that a group approach to interdependency assessment is probably the most fruitful one.

7.3.4 Summary.

This exercise has indicated clearly that there are immediate design benefits to be had from the SDM requirements preparation and interdependency analysis activities. The common source of these benefits lies partially in the simple fact of having to think carefully, and repeatedly, in a structured way, about what each requirement really means, about how each might be implemented, and about how alternative implementation schemes interact. In the next section we analyze the architecture for the new Budgeting System that emerges from this analysis.

7.4 AN ARCHITECTURE FOR M.I.T.'S NEW BUDGETING SYSTEM.

Once the interdependency analysis has been completed, the interdependency statements can be entered into the computer for use in the decomposition analysis. The Budgeting System interdependency statements are of the form:

node1	node2	weight	description
-------	-------	--------	-------------

The weight values are entered as 'W', 'A', or 'S', as discussed earlier. The "description" is a brief text commentary used to document the rationale underlying the designers' assessment of the existence of that particular interdependency. A complete listing of the Budgeting System interdependencies is given in Appendix K.

It should be noted that the capability of entering, storing, and retrieving interdependency descriptive information was judged by Andreu to be an important feature not present in his initial version of the SDM analysis package. While the techniques that have been developed for this purpose in the current effort have been found useful, there are some further improvements that could be made, and are discussed in the final section.

7.4.1 Decomposition Analysis Results.

The Budgeting System interdependencies define its requirements graph. The interdependencies data file (Appendix K) was converted, using a simple standalone routine (to be incorporated into the analysis package in the future), to another data file (containing no text information) that could then be used as input to the MASTER decomposition routines. These routines are described and documented in Appendix E.

The facilities of the analysis package were then used to develop decompositions of the requirements graph, to evaluate them using the objective function M , to modify and manipulate the decompositions in various ways, and to save and print out the results so obtained.

Each of the five decomposition techniques (four clustering techniques, and the interchange algorithm) were applied to the Budgeting System requirements graph. The outcomes are shown in Table 7.1. Of the four clustering methods, HIER3 produced the best overall decomposition, with an objective function value of $M = 0.67$. The objective function values for HIER1, HIER2, and HIER4(19) were, respectively, 0.05, 0.27, and 0.27.

(19) These algorithms are discussed in detail in Chapter 5.

<u>Algorithm</u>	<u>Objective Function Value for Best Located Decomposition</u>
------------------	--

HIER1	0.05
HIER2	0.27
HIER3	0.67
HIER4	0.27
INTERCHANGE	0.85

Table 7.1

**Comparison of results of five decomposition
algorithms on the Budgeting System requirements graph**

The interchange algorithm was also applied to the requirements graph, and produced a decomposition with $M = 0.85$. This decomposition, then, was judged to be the best in terms of identifying high-strength, low-coupling subgraphs (as measured by M). This best decomposition of the requirements graph produced by the interchange method is illustrated in Figure 7.2. Appendix L contains a listing of the abbreviated Budgeting System requirements (no Comment sections included there, for brevity) organized according to subgraph. Finally, Appendix M contains a listing of the inter-requirement links between each identified pair of subgraphs.

The task that remains, then, is to study the decomposition - both the requirements subsets, and the sets of interdependencies between requirement subsets - so as to formulate an interpretation of the graph decomposition as a system architecture. At the same time we seek to identify anomalies, counterintuitive results, etc., that might indicate earlier errors in assessments, requirements formulation, etc. Alternatively, anomalous results might turn out, on closer inspection, to be correct after all, but simply unforeseen. Such issues will be examined in greater detail in the next section.

Subgraph

Requirements

1	7,28,38,56,57-62,65,66,68,71,76
2	18-26,29,31-34,36,39-42
3	16,43-52,64,74
4	15,77
5	9,10,13
6	53-55,67,69,70
7	11,12,14
8	5,6,27,35
9	8,63,75
10	1-4,17,30,37
11	72,73

Figure 7.2

Best located decomposition produced by the interchange method on the Budgeting System requirements graph.

7.4.2 Analysis of Design Subproblems.

A total of eleven design subproblems were identified in the best decomposition of the requirements graph. Three of these subproblems are "middle-sized," containing 15, 19, and 13 requirements; the remainder are somewhat smaller, ranging in size from two to seven requirements each.

Figure 7.3 shows the relationships between the eleven design subproblems and the 21 inter-subproblem linkages. The size of the linkages shown in the figure is related to the number of interdependencies encompassed by each, as explained in Section 7.3. This figure will be expanded with additional descriptive information following our discussion of the individual subproblems and linkages in this and the next section.

Subproblem 1 - Preparation of Budget Proposals.

This subproblem centers on the preparation of budget proposals. One of the intended features of the new Budgeting System is a much more streamlined, easier-to-use set of proposal preparation facilities, including on-line preparation, automatic checking and cross-checking of entered data for consistency and reasonableness of values, on-line examination by the Budget Office, and on-line modification by the budget preparers (administrative officers, department heads, etc.). Most of the requirements in this subproblem hinge directly on these related activities.

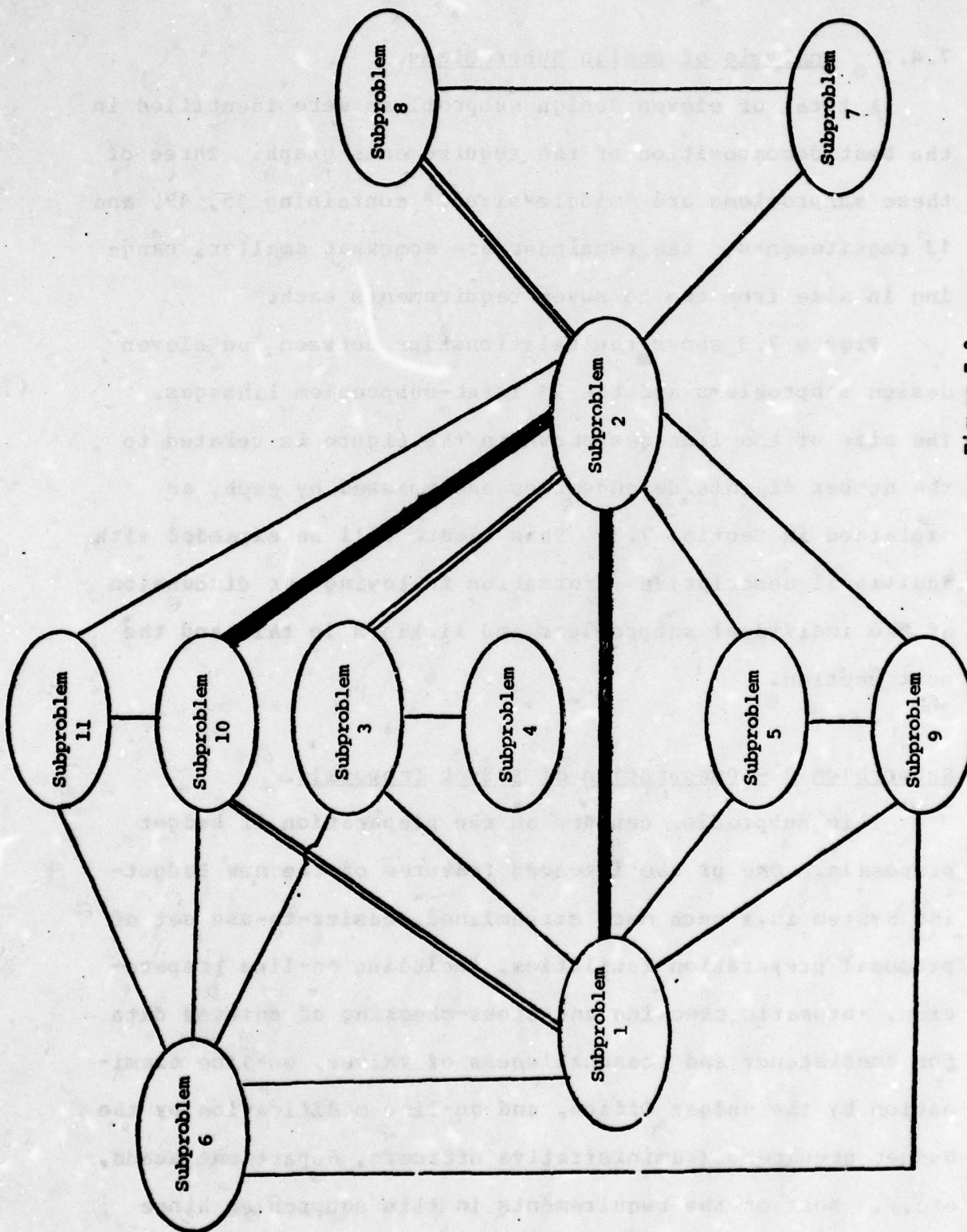


Figure 7.3
Relationships among the eleven design subproblems

Requirements 7, 28, 38, 62, and 68 all are related to the maintenance of various kinds of data directly relevant to proposal preparation. The fact that these data sources are identified together is useful for later detailed design of the Budgeting System database - e.g., when deciding on segment structure, record layouts, etc., it is most useful to have a clear idea of what data is most likely to be used jointly or in closely related activities.

Requirements 56 and 57 pertain to the proposal preparation process itself. Requirements 58, 59, 60, and 61 all pertain to the issues surrounding the checking, editing, and revision of budget proposals or changes to pending proposals. Requirement 71, regarding fund draft checking, is closely tied to various other requirements within this subproblem, including those involved with special financial arrangements (7,68), and those requirements with similar processing steps (59,60,61,62).

There are two seeming anomalies, in the presence of requirements 65, 66, and 76 in this subproblem. A deeper examination, however, reinforces the correctness of this assignment. Requirements 65 and 66 involve handling of research proposals by the Budgeting System. Research proposals may not appear at first glance to have much in common with budget proposals. However, as discussed in Section 7.3.3 earlier, one of the useful discoveries made by the

chief designer in the course of the SDM interdependency analysis was the existence of strong potential implementation parallels between the handling of research and budget proposal preparation. (20) These parallels manifest themselves in interdependencies that eventually result in the research proposal preparation requirements being grouped together, for design consideration purposes, with the budget proposal preparation requirements.

The existence of requirement 76 in Subproblem 1 similarly makes good sense upon closer examination. The Institute's overhead recovery rate is, in fact, a key item of information in budget proposal preparation. The rate is adjusted as a function of the Institute's financial situation each year. The intention in the new Budgeting System is to estimate the rate for the coming fiscal year on the basis of information available in the budget proposals (hence requirement 76) and make this estimate available to the budget officers for their proposal preparation activities. Thus the manner in which recovery rate calculations are made is closely tied in with proposal preparation, so should be considered together for design purposes.

(20) specifically, the use of a common suspense file approach pending final acceptance of the proposals.

Subproblem 2 - Operations Reporting.

The second subproblem is the largest in terms of the number of requirements included: 19 requirements. Its central focus might be termed "operations reporting." Basically, this subproblem addresses monitoring of actual income and expense information against the operating budget - i.e., the control side of the budgetary process. Since this is perhaps the largest and most important function to be provided by the new Budgeting System, it is most appropriate that it should also turn out to be the focus of the largest design subproblem.

Certain of the requirements in this subgroup directly address the operational analysis and reporting capabilities of the new system, including requirements 18, 19, 29, 31, 33, and 39. Many of the remaining requirements ended up in this group because of strong data interrelationships between them and the ones cited above. This includes requirements 20 through 26. These requirements specify that certain databases will be maintained by the Budgeting System, databases intimately linked to the provision of the monitoring and reporting functions to be provided.

It is interesting to note that these requirements end up together in the requirements decomposition because of their data-oriented interrelationships, as opposed to their processing interrelationships. One of the recent "discover-

ies" in software engineering research concerns the frequently underestimated importance of the role of data structures and data handling in system design. Earlier work usually assumed program control flow to be the pre-eminent concern, data organization to be of secondary importance; more recent work has tended to elevate the relative importance of data organization (Jackson 75). The evidence that emerges from the present study is that SDM is inherently quite compatible with this more balanced view of the importance of both processing and data interrelationships in determining good system structure.

Finally, a few other requirements fall into this subproblem because they specify reporting needs that would be met primarily using data common to other requirements of the same subproblem. Included here are requirements 34, 36, 40, 41, and 42. All five of these statements refer to potential reporting requirements of various types that all would most likely involve budget and actual operational data common to other requirements in this subproblem.

In summary, while Subproblem 2 is a fairly large subproblem, careful study indicates that the 19 requirements do "hang together" for design purposes, largely as a result of their common data implications.

Subproblem 3 - Database Access for Nonstandard Report Generation.

The third subproblem contains 13 requirements. The focus of this subproblem is database access for purposes other than standard report generation. This includes requirements for users' ad-hoc access (requirements 44 and 45), users' access via the report writing facility (requirements 43 and 46), and access to the Budgeting System's databases via other systems (requirement 16). This subproblem also includes certain database security requirements that pertain to user access, namely, requirement 47 through 52. These requirements all relate to data ownership and data element controls that are closely related to data ownership. Since these security issues manifest themselves, in implementation terms, primarily at the point of data access, it makes very good sense that they be grouped together with other data access requirements.

The presence of requirement 74 in this subproblem also makes good sense: formal training issues would undoubtedly be heavily concerned with data access, as this would be the main interest of most users. An interesting side point regarding this requirement is that, at first glance, it may not appear to be a design-relevant issue at all. This question was in fact debated among the system designers and the SDM researcher, with (eventually) the opposite conclusion.

The main argument ran as follows. There is no requirement stating that the system (specifically, access to data) be "easy for users to use." Such a requirement would be too general, at too high an abstraction level, to be appropriate for SDM analysis. In contrast, requirement 74, specifying the need for formal user training, is more specific, at an abstraction level comparable with the other requirements, and at the same time achieves most of the same results. In thinking about how one might "implement" a requirement such as 74, one is led to imagine what a trainer would have to say to explain various aspects of system functioning thought to be of interest to different user groups. Thinking in this way leads the system designers to adjust their conceptual models of implementation for the associated requirements, with an eye to the need for formal training.

Finally, requirement 64 (support for current budgeting techniques) also ended up in Subproblem 3. The case for this requirement being in this subproblem is not obvious initially. However, a review of the "audit trail" of interdependency assessments indicates that requirement 64 was found to be interdependent with only two other requirements: weakly with 32 (Subproblem 2), and with average strength with requirement 74 (Subproblem 3). Its ending up in the present subproblem is therefore justifiable on purely "mechanical" grounds. The rationale for its interdependency

with 74 concerns user training. Formal training on the one hand, and support for all budgeting methods on the other, are seen to be discordant requirements. Through this line of reasoning one can envision second-order relationships between 64 and many of the other requirements in Subproblem 3, bearing on the fact that a multitude of budgeting approaches would probably make the implementation of user access to data, and control of data, more difficult to achieve. Clearly, users may require access to different data combinations under, say, line item budgeting than they would under program budgeting. However, the fact remains that requirement 64 is only tangentially concerned with the main focus of Subproblem 3, but as it is no "closer" to any other subproblem, it ended up there.

Subproblem 4 - Physical Report Handling.

This subproblem only contains two requirements, and is therefore rather easy to interpret. Its focus is physical report handling. One requirement (15) concerns the report medium, specifying that reports will be physically easy to handle. In implementation terms, this may be viewed as arguing against the production of reports on standard 11x14 inch computer paper, and suggests the use of 8 1/2 x 11 inch paper for reports. In fact, this requirement is an example of a situation discussed earlier: the designers' original

requirement statement (see Appendix H, requirement 7) actually specified the way in which this requirement could be achieved, i.e., contained within it its own implementation approach. In the spirit of SDM analysis, the requirement was re-written so as to be functional in form, and implementation-free. However, as the designers did not want to lose sight of their implementation concept (and indeed wanted other readers of the requirement statements to be aware of it also) they decided to keep it, in the form of a comment on requirement 15 (Appendix I).

The other requirement in this subproblem, 77, simply says that the new Budgeting system should be designed so as to minimize unnecessary delay in report production and distribution. A discordant interdependency between requirements 15 and 17 occurs because one way of meeting 77 would be to employ a number of RJE terminals or remote printers at user sites; but this would likely result in reports being printed on large forms, thereby counteracting requirement 15.

Subproblem 5 - Use of EFT to Define Personnel Levels.

Subproblem 5 contains three requirements, and is straightforward to interpret. The central focus for this subproblem is the use of EFT ("effective full-time") units for dealing with personnel data. The idea is that many

administrative officers (AO's) find it easier to think in terms of EFT units for personnel budgets than in dollar terms, hence requirement 9 specifies that EFT may in fact be employed to develop personnel budgets. However, as budgets are necessarily framed eventually in dollar terms, there must be a mechanism within the Budgeting System for converting between EFT and dollars. This is not as simple a task as it may at first appear; a number of detailed issues have to be considered, and the conversion algorithms may be rather complex.

Finally, requirement 13 specifies the need for flexibility in manpower reporting (as opposed to budgeting). In practice it is easier to report some types of manpower in man-months, other types in, say, man-hours, etc. Clearly, allowing such reporting flexibility complicates still further the dollars-EFT conversion issue, hence this requirement's presence here.

Subproblem 6 - Maintenance of Database Integrity, Control.

This subproblem includes six requirements, and has as a central focus the maintenance of database integrity and control. Requirements 53, 54, 55, and 67 all relate directly to this issue in various ways. It should be noted that these requirements impact system design primarily in the choice of a commercial DBMS to be used as the heart of the

new system, as there is no intent in the minds of the designers to develop their own underlying data management software.

Requirements 69 and 70 pertain to computerized funds drafts being performed, either on-line or via batch transactions. Once again, these requirements might appear to be rather distant from the preceding four. Closer examination dispels this notion, and confirms again the correctness of the partitioning. Funds drafting, and the checking and verification thereof, is perhaps the most sensitive data-handling aspect of the new system, as it deals directly with the movement and control of real, spendable credits.(21) Therefore the requirements to allow electronic funds drafting in the new system are especially closely related to the integrity concerns, notably the audit trail requirement (67). In implementing the integrity requirements, very close attention will also have to be paid simultaneously to the funds drafting issue.

The separation of requirements 69 and 70 from other requirements dealing with funds (see Subproblem 10) is another good example of the issue discussed on page 373, concerning the need to carefully differentiate between requirements that are logically related (e.g., all require-

(21) most of the system's databases will consist of budget data, or else real expense (credit) data.

ments dealing with funds), and those that are related in implementation terms.

Subproblem 7 - Organization of Budgeting/Accounting Objects.

This subproblem includes three requirements. The subproblem's focus is the organization of the budgeting/accounting "objects." This term has an unambiguous meaning to the budgeting and administrative staff of the Institute. "Objects" represent one way of organizing the elements of income and expenditure of the Institute; for instance, "secretarial salaries" might be an expenditure object, while "federal grants" might be a revenue object. In the present budgeting system, there is a fixed set of objects, and codes for each object, with which all concerned must work. (22)

The new system, as these requirements indicate, is to have greater flexibility in terms of object definition. Specifically, it will be possible to define multiple hierarchies of objects. Also, departments will be able to define their own personal objects within the central system.

The three requirements of this subproblem are clearly related quite closely for implementation purposes. In particular, it may be noted that the requirements for multiple

(22) at least as far as the central budgeting office is concerned, although many departments also run their own parallel systems that better meet their specific needs.

object hierarchies (11 and 12) probably rule out the possibility of using a hierarchically-oriented commercial DBMS such as IBM's Information Management System (IBM 74) for the system's database manager.

Subproblem 8 - Development and Management of Planning Data.

This subproblem contains four requirements, and has as its focus the development and management of Institute intermediate-range planning data. Planning data differs from budget data in a number of ways. First, its development is neither homogeneous nor universal across the Institute. Some departments develop much more, and more highly detailed, planning data than do others; some plan up to two years ahead, others up to three years, still others five years out. Also, the nature of the data differs from budget data, often being more in the form of descriptions of underlying goals, directions, etc., for a given department, rather than hard numbers at a relatively fine level of detail.

During their analysis of the present planning/budgeting practice, the Budget System designers found a need for some level of automated assistance to the planning function of various administrative staff. Since this function is fairly closely related to the budgeting function, to include some planning assistance capabilities in the new system seemed

appropriate. Requirements 5 and 6 address the planning data issue directly. Requirement 35 concerns data requirements for the "Dynamic Model" - M.I.T.'s financial forecasting model. Since many of the model's data requirements are in the nature of planning data, it is appropriate that this requirement fall in the current subproblem. Specifically, the determination of the type of planning data to be obtained from the Institute managers ought to be influenced directly by the data needs of the model.

The inclusion of requirement 27 in this subproblem is another instance of a surprisingly intelligent outcome of the SDM decomposition. While logically related (in the sense discussed on page 373) to requirements 21 through 26, this requirement differs on one very important respect: historical actual data is the key database referenced by managers and administrators in drawing up their intermediate-range plans. None of the other requirements 21-26 have this property. The implementation of planning assistance facilities in the new Budgeting System must take into consideration both what planning data will be captured, and what data will be required by managers in developing their plans. Requirements 5 and 6 correspond to the former concern, requirement 27 to the latter.

Subproblem 9 - Employee Benefit Rate Calculations.

This subproblem, with three requirements, concerns employee benefit rate calculations. In the present budgeting system, management of employee benefits may present some complexities, especially when a) ...; b) ...; or c) For instance, a department's budget may be approved given a certain staffing profile. That profile may then be altered through substitution of certain staff members for others (e.g., increasing the number of individuals assigned to a particular research project). This kind of change may require a compensating change in employee benefit amounts being charged to that department. Such a change is, at present, frequently "lost in the shuffle." Subproblem 9 addresses the need for an effective mechanism for controlling the employee benefit issue within the Current and Future budgets.

Since employee benefit amounts are a percentage of personnel salaries and wages, it makes good sense that requirement 8 be included in this subproblem. Similarly, requirement 75 is concerned with handling non-standard employee benefits (e.g., benefits for a part-time faculty member), which are subject to very similar needs for control as are standard benefits.

Subproblem 10 - Organization and Management of Fund Accounts.

There are seven requirements in Subproblem 10. The focus of this subproblem is the organization and management of fund accounts. At the heart of this design subproblem is requirement 30, "...facilitate the effective use of funds accounts." The other requirements in this subproblem (with one exception, to be explained shortly) all partially derive from requirement 30. Requirements 1, 2, and 3 all specify alternative ways in which fund data may be organized (and hence retrieved during queries, etc.). Requirement 4 addresses the need to supply adequate descriptive information for each fund account within the database itself. Requirement 37 pertains to reporting standard fund information. Finally, requirement 17 specifies the need for access to data in the databases of other Institute DP systems. The system designers saw this as closely associated with effective fund management, largely because much of the information that would be needed in order for requirement 30 to be properly implemented resides in the Gift System.(23) A direct link to the latter system would be superior to duplicating and separately maintaining the necessary databases.

(23) This is the system that is used to manage gifts, bequests, etc.

Subproblem 11 - Database Expandability.

The final subproblem contains two requirements, and its focus is database expandability. Specifically, requirements 72 and 73 state that, unlike the present budgeting system, the new system will allow new data elements to be defined within objects or accounts, in order to provide additional flexibility and usefulness for the system's users. For instance, one department may wish to summarize a certain set of account balances in a unique way. The new system would allow a special data item to be defined to hold the appropriate summary data, and also to tie the new item logically to the lower-level items it summarizes, so that when changes are made to the lower-level items the summary item will also be updated. (24) This requirement ties into the concept of logical data independence that has come to prominence along with the use of database management systems (Martin 77).

* * * * *

This concludes the analysis and interpretation of the eleven identified system design subproblems. The subproblems and their interpretations are summarized in Table 7.2. In the next section we analyze the interrelationships (sets of interdependencies) between the various subproblems. The combined interpretation of subproblem and interrelationship

(24) either actually or virtually - see (Folius, et. al. 76).

Subproblem

Summary Description

- 1 Preparation of budget proposals
- 2 Operations reporting
- 3 Database access for purposes other than standard report generation
- 4 Physical report handling
- 5 Use of EFT to define personnel levels
- 6 Maintenance of database integrity, control
- 7 Organization of budgeting/accounting objects
- 8 Development and management of planning data
- 9 Employee benefit rate calculations
- 10 Organization and management of fund accounts
- 11 Database expandability

Table 7.2

Subproblem Summary Descriptions.

analyses constitutes the architecture interpretation for the new Budgeting System. The final section contains concluding comments pertaining to global aspects of this architecture.

7.4.3 Analysis of Subproblem Interrelationships.

There is a total of 20 links interconnecting the 11 design subproblems in the new Budgeting System architecture. Some statistics regarding these links are shown in Table 7.3 below. It is shown there that the average number of interdependencies per subproblem link is 4.5; however, there are only five links consisting of more than five interdependencies. Since both number of interdependencies as well as interdependency weights are important determinants of link strength, Table 7.3 also shows total weight for each link. This is just the sum of the weights on all the interdependencies making up each link. From this table it may be seen that the distribution of link total weights is as shown in Figure 7.4 following. From the figure, it is clear that the design partitioning has two rather strongly interconnected subproblems ((1,2), and (2,10)), three subproblem linkages of medium strength ((1,10), (2,3), and (2,8)), while the remaining subproblem linkages have a total weight of less than 2.0, so are relatively weakly connected. In the diagram of Figure 7.3 earlier, the strongest linkages are shown shaded, the medium-weight ones are drawn as double lines, while the remainder are shown as single lines.

<u>ID #</u>	<u>First Subproblem</u>	<u>Second Subproblem</u>	<u>Number of Linking Interdependencies</u>	<u>Total Weight</u>	<u>Average Weight</u>
1	1	2	20	7.6	0.38
2	1	3	5	1.9	0.38
3	1	5	4	1.1	0.28
4	1	6	3	1.2	0.40
5	1	9	4	1.4	0.35
6	1	10	10	3.8	0.38
7	2	3	7	3.2	0.46
8	2	5	2	1.0	0.50
9	2	7	2	1.3	0.65
10	2	8	10	3.8	0.38
11	2	9	2	1.0	0.50
12	2	10	16	6.8	0.43
13	2	11	5	1.9	0.38
14	3	4	1	0.5	0.50
15	3	6	2	0.7	0.35
16	5	9	2	1.3	0.65
17	6	9	3	0.9	0.30
18	6	10	3	0.9	0.30
19	6	11	1	0.5	0.50
20	7	8	1	0.5	0.50
21	10	11	1	0.8	0.80

Table 7.3

Statistics for the Inter-subproblem Linkages

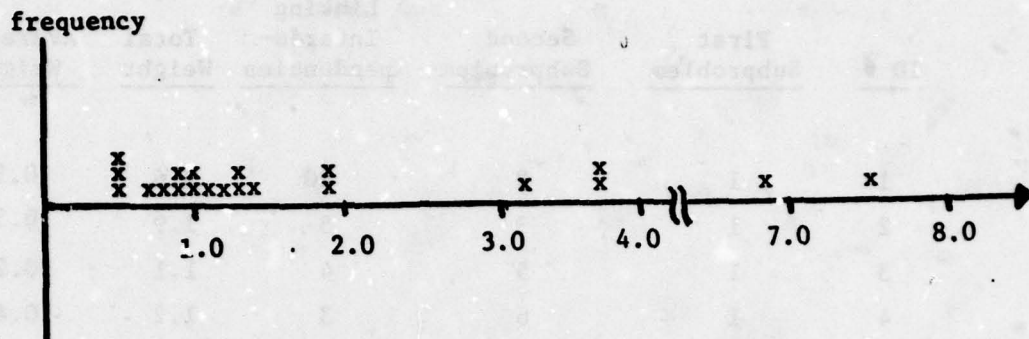


Figure 7.4

Distribution of Link Total Weights.

It is worth recalling at this point that the underlying motivation for this entire SDM exercise is to formulate a system architecture which exhibits high module strength and low coupling. The objective function M , of course, is a formal attempt to quantify that concept. Informally and judgmentally, a system decomposition with only two relatively strongly interconnected subproblem pairs, three pairs with medium interconnection strength, and fifteen with relatively weak interconnection strength should probably be judged as a fairly good one from this point of view.

We now examine each subproblem linkage, and describe an interpretation of the nature of, and the reasoning behind, each.

Linkage 1 (Subproblems 1 and 2).

This is the largest linkage, with a total link weight of 7.6. Eleven of the linking interdependencies represent common data issues: databases defined, either implicitly or explicitly, in conjunction with requirements in Subproblem 2 that are also needed for effecting certain proposal preparation-related requirements in Subproblem 1. This includes in particular Chart of Accounts data, and Current and Future Budget databases.

The remaining linking interdependencies represent concordant relationships that arise because of common processing techniques. In one case, the automatic proration feature may be used to good effect in proposal preparation as well as operations monitoring activities; in the other case, potential methods of operational monitoring facilitate certain aspects of monitoring proposal preparation.

Linkage 2 (Subproblems 1 and 3).

This linkage has a total weight of 1.9, and includes five interdependencies. Two of the interdependencies represent the need for training users to use the system for proposal preparation. The other three represent the use of the menu-oriented query facility for proposal and fund draft review and checking.

Linkage 3 (Subproblems 1 and 5).

This linkage consists of four interdependencies, with a total weight of 1.1. The common focus for all four concerns the ability of users to prepare and/or monitor the personnel component of budget proposals in the most convenient units (typically, EFT or dollars).

Linkage 4 (Subproblems 1 and 6).

This linkage includes three interdependencies, with total weight 1.2. The process of preparing budget proposals (Subproblem 1) requires administrators to access various kinds of data that will, in the future, be available via the new system. The focus of this linkage concerns the implementation issues surrounding protecting the security and integrity (Subproblem 6) of the data that will be accessed for proposal preparation purposes.

Linkage 5 (Subproblems 1 and 2).

This linkage includes four interdependencies, and has a total weight of 1.4. These interdependencies all pertain to the role of employee benefit calculations in the proposal preparation process.

Linkage 6 (Subproblems 1 and 10).

This linkage contains 10 interdependencies, and has a total weight of 3.8. Subproblem 1 addresses budget proposal preparation generally, and the requirements within Subproblem 1 that connect to Subproblem 10 are concerned specifically with the role that funds play in proposal preparation. Subproblem 10 focuses on the organization and use of fund accounts. Good information is the key to better management of Institute funds (gifts, bequests, etc.). At the present time fund monies are frequently not used to their greatest benefit, because individuals who make expenditure decisions haven't been informed of, and have no easy way of discovering, the existence of certain funds whose designation meets their needs. The intention in the new Budgeting System is to make fund purpose information readily available to users, and to otherwise orient the reporting and control of fund data so as to make more effective use of fund monies. This would conserve general monies to more fully meet the needs to which funds do not apply.

Linkage 7 (Subproblems 2 and 3).

This linkage contains seven interdependencies, with total weight of 3.2. All seven interdependencies have a fairly strong common focus: they all represent techniques to allow users to access data in a manner other than via standard reports (special reports, ad hoc queries, etc.).

Linkage 8 (Subproblems 2 and 5).

There are two interdependencies making up this linkage, with a combined weight of 1.0. The focus of these interdependencies is the proration of personnel budgets for the production of periodic operating reports.

Linkage 9 (Subproblems 2 and 7).

This linkage includes two interdependencies, with a combined weight of 1.3. It focuses on the use of a hierarchical organization of object codes for facilitating the production of special reports.

Linkage 10 (Subproblems 2 and 8).

This linkage contains ten interdependencies, with a total weight of 3.8. All of these interdependencies have a clear common focus, namely, data commonality between requirements for operations reporting (Subproblem 2) and for planning (Subproblem 8). Although as pointed out earlier, planning data and budgeting data is not identical, there is enough commonality to generate numerous implementation-level interdependencies. For instance, some of the data required by the Dynamic Model (requirement 35 in Subproblem 8) may be obtained from various managers' Future Budgets (requirement 22, Subproblem 2) databases.

Linkage 11 (Subproblems 2 and 9).

This linkage contains two interdependencies, with total weight 1.0. Its focus is the data management issues common to Future Budget personnel data.

Linkage 12 (Subproblems 2 and 10).

This linkage contains 16 interdependencies, with a total weight of 6.8. All of the interdependencies within this linkage concern different aspects of data access commonality between the two subproblems. Eight of the interdependencies focus on databases common to ad hoc retrieval requests associated with operations monitoring, and similar requests associated with funds management. Another four interdependencies are related to databases common to operations monitoring, and to making effective use of fund accounts. Another three relate to similar databases common to standard report generation for operations monitoring and for fund management. Finally, one of the interdependencies represents the common need for access to other systems' data files.

Linkage 13 (Subproblems 2 and 11).

This linkage represents five interdependencies, with a total weight of 1.9. All five interdependencies pertain to the application of the facility for adding new data item

types to currently existing databases, to the development of operations monitoring requirements.

Linkage 14 (Subproblems 3 and 4).

There is a single interdependency in this linkage, with a weight of 0.5. The focus of this link is mechanisms for speeding the delivery of Budgeting System information, in the form of standard reports, to system users.

Linkage 15 (Subproblems 3 and 6).

There are two interdependencies within this linkage, with a total weight of 0.7. Their common focus is the maintenance of database integrity and security by means of a transaction logging technique.

Linkage 16 (Subproblems 5 and 9).

There are two interdependencies in this linkage, with a total weight of 1.3. Their common focus is the conversion of personnel data between dollars and EFT.

Linkage 17 (Subproblems 6 and 9).

This linkage includes three interdependencies, with combined weight 0.9. Their focus is audit trail maintenance in the face of automatic system updating of certain data items.

Linkage 18 (Subproblems 6 and 10).

This linkage includes three interdependencies, total weight of 0.9. They all focus on the effecting of computer-based funds drafting.

Linkage 19 (Subproblems 6 and 11).

This linkage consists of a single interdependency, total weight 0.5. The linkage concerns the implementation of a data change log in a restructurable database.

Linkage 20 (Subproblems 7 and 8).

This linkage contains a single interdependency, with a weight of 0.5. It concerns the use of the object hierarchy for organizing the development of planning data for the Dynamic Model.

Linkage 21 (Subproblems 10 and 11).

The final linkage also includes but one interdependency, with a weight of 0.8. Its focus is the addition of fund purpose categorization information to the funds accounts.

* * * * *

This completes the description of the individual linkages between the design subproblems. Summary descriptions of the 21 inter-subproblem linkages are given in Table 7.4.

<u>ID</u>	<u>Subgraphs</u>	<u>Summary Description</u>
1	1,2	common databases; common processing viz. proration, monitoring.
2	1,3	training in proposal prep. use; proposal/ fund draft review and checking via query.
3	1,5	alternative units for personnel data.
4	1,6	protection of the security and integrity of proposal preparation data.
5	1,9	employee benefit calculations in pro- posal preparation.
6	1,10	effective use of funds in proposal preparation.
7	2,3	data access for nonstandard usage.
8	2,5	personnel data proration in periodic rpts.
9	2,7	use of hierarchical object organization in production of special reports.
10	2,8	operations reporting and planning data commonality.
11	2,9	common Future Budgeting/Personnel Budgeting data management issues.
12	2,10	data commonality: ad hoc retrieval, monitor- ing standard reporting, other systems.
13	2,11	extension of operations monitoring data files.
14	3,4	prompt report delivery.
15	3,6	use of logging to effect database integrity and security.
16	5,9	conversion of personnel data: dollars vs. FRT.

17	6,9	maintenance of audit trail in face of auto. updating of data elements.
18	6,10	effecting computer-based funds drafting.
19	6,11	maintenance of change log when database structure is modified.
20	7,8	hierarchy of object codes used to effect generation of Dynamic Model data.
21	10,11	addition of fund purpose categorization data to fund accounts.

Table 7.4

Summary Description of the Subproblem Linkages.

Figure 7.5 shows a complete description of the SDM-derived architecture for the new Budgeting System. Each design subproblem and linkage is labelled in an abbreviated fashion, based on the descriptions given in Tables 7.2 and 7.4.

In the final section of this report we briefly discuss certain broad issues that arise out of this architecture. We also summarize there the work to date and suggest some areas for further research.

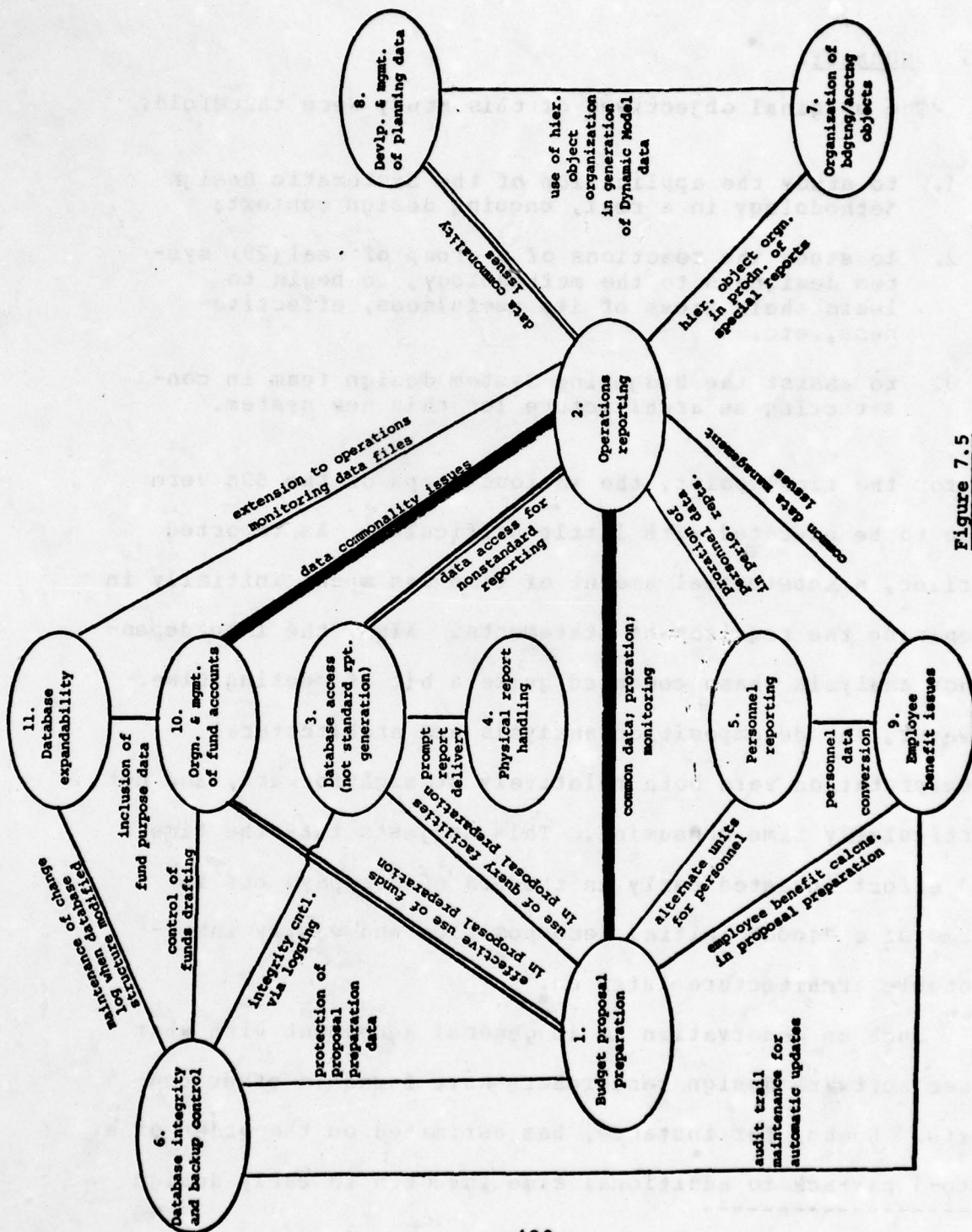


Figure 7.5

Complete description of the SDM architecture for the new Budgeting System

7.5 SUMMARY.

The original objectives of this study were threefold:

1. to study the application of the Systematic Design Methodology in a real, ongoing design context;
2. to study the reactions of a group of real (25) system designers to the methodology, to begin to learn their views of its usefulness, effectiveness, etc.
3. to assist the Budgeting System design team in constructing an architecture for this new system.

As for the first point, the various steps of the SDM were able to be executed with little difficulty. As reported earlier, a substantial amount of time was spent initially in preparing the requirement statements. Also, the interdependency analysis phase consumed quite a bit of meeting time. However, the decomposition analysis and architectural interpretation were both relatively straightforward, and not particularly time consuming. This suggests that the time and effort invested early in the SDM effort pays off in terms of a "good" initial decomposition and easily interpretable architecture later on.

Such an observation is in general agreement with what other software design researchers have found in other contexts. Boehm, for instance, has estimated on the order of a 3-to-1 payback to additional time invested in early design

(25) as opposed to SDM researchers playing the role of system designers

activities (Boehm 75). Also, the truth of this observation in the SDM context is verified by earlier applications carried out by other SDM researchers. Andreu (Andreu 77(a)) applied SDM to the design of a database management system. He adopted a set of government-issued DBMS requirement statements for use in his design. His first pass at building a system architecture resulted in a rather unsatisfactory decomposition, with a few large, unwieldy, hard-to-interpret subproblems. After "completing the requirements set" by studying the requirements statements carefully for missing requirements, ambiguous statements, etc., and making a number of additions and modifications, a second decomposition resulted in what Andreu argued was a much better architecture - smaller, more coherent subproblems arranged in a fashion he found much easier to interpret.

In contrast, in the present study we spent much more time in framing and refining the original requirement statements. In fact, the requirement specifications upon which the SDM statements were based was the result of over a year of study, analysis, interviews, etc. Also, the SDM version of the statements was given reasonable in-depth study over a number of iterations by both the system designers and the SDM researcher. It is most likely for this reason that the requirements decomposition and resulting architecture turned out to be as good as it did after a single "pass" of the SDM

analysis. With very few exceptions (to be discussed below) the design subproblems and linkages were clear and easy to interpret. All subproblems were found to have an obvious design focus, as described earlier. Similarly, the important design implications of the various inter-subproblem linkages were easily identified. Judged by this mixture of intuitive and explicit measures, SDM functioned well in guiding us to the identification of a good architecture for the new Budgeting System.

The second point concerns reactions of the Budgeting System designers to SDM. They expressed both positive and negative reactions to the analysis exercise, all of which were discussed earlier (see Section 7.3.3). In summary, the main negative reactions concerned the time required for the analysis, and some uncertainty about the overall value of the exercise (the latter occurred mostly at the outset). Both issues were tempered, of course, by an appreciation of the research nature of the study. The positive reactions concerned new design ideas, as well as clarification and improvement of current ideas, that emerged during the exercise; discovery of new ways of approaching the design task in general (e.g., separation of functional concerns from implementation issues); and their belief that the final architecture would be of assistance in the later detailed design efforts.

As for the final point, the eventual value of the Budgeting System architecture presented in the preceding section cannot be known at this time. Rather, it will be necessary for the SDM researchers to follow up this exercise in the future to learn what kind of impact this study might promulgate.

The study has provided a number of other insights, many of which were mentioned earlier in this report. The most important of these are summarized below.

1. The design architecture that emerged from our work did prove to be relatively "clean" (high strength, low coupling). However, a few minor points might bear additional investigation by the system designers. For instance, requirement 64 was found not to have an obvious "home" in any subproblem. This may suggest that either certain interdependencies between it and other requirements were missed during the earlier analysis, or possibly that other requirements more closely associated with requirement 64 are missing from the requirement set.
2. A second minor decomposition issue concerns the size of Subproblem 2. As discussed in Section 7.4.2 earlier, there are good reasons for the relatively broad scope of this subproblem. The

operations reporting function is central to the Budgeting System, hence we should expect this subproblem to encompass a larger collection of requirement statements than many of the others. The centrality of the subproblem is further evidenced by noting that it has eight linkages to other subproblems - substantially more than any of the others (see Figure 7.5). The centrality of this subproblem should serve as a signal to the system designers that perhaps the detailed design ought to also center on the implementation of these requirements. Also, in general the presence of a large subproblem such as this in a decomposition should be carefully studied, as it may suggest possible improvements to the decomposition. Andreu found the occurrence of especially large, heterogeneous subproblems to be "caused" by incomplete or poorly framed requirements set. Also, it could be that certain interdependency assessments are missing or in error. While they will not be further investigated here, these possibilities are worth some study by the system designers in the future.

3. A third summary point is that the SDM techniques used in this study seemed to function rather well. The use of a three-way breakdown for specifying interdependency weights should be judged quite effective in practice: a threefold distinction could be made easily by the designers, and the need for finer detail rarely arose. Other mechanisms - the interdependency data form, use of WSCRIPT to manage the requirement statements, etc. - worked well. The decomposition package also functioned effectively. A number of ideas for marginal improvements to the package arose in the course of its application to the Budgeting System problem, but these will not be elaborated on here.
4. One lesson that came through quite clearly in this exercise is the important role data linkages play in determining requirement interdependencies in this kind of system design. In a number of cases, part or all of the common implementation issues tying a set of requirements together within a subproblem were common databases required for their implementation. Also, such data commonality formed the basis of the linkage between various subproblems. It is interesting to note that in

the previous SDM application exercises this important role of data commonality was not evidenced. This is not too surprising since systems like the Budgeting System are much more concerned with capturing, processing, reporting, and otherwise dealing with various databases than would be "systems" software such as a DBMS or an operating system, at least at the architecture level. However, while perhaps not so surprising in retrospect, this observation should serve to make the importance of database organization for the Budgeting System stand out during future design and implementation work.

* * * * *

While still a research project, the Systematic Design Methodology is proving its effectiveness in aiding system architects to organize and manage the many and diverse requirements typical of complex systems design. This study has suggested new improvements that may be made to the methodology, while again confirming its fundamental soundness and value.

In the next chapter, we examine this issue - the fundamental efficacy of the Systematic Design Methodology - in more detail.

Chapter VIII

SDM EFFICACY.

8.1 INTRODUCTION.

The growth in the size and complexity of computer software systems during the last decade has made it increasingly important for software developers to create and use various new tools and techniques to assist them in managing the complexity of their task. New techniques have been developed to assist and guide system developers through all stages of the development cycle - from very early stages (e.g., users' needs analysis methods such as TRACE (Altman, et. al. 71)), through design (Structured Design (Myers 78)), coding (Structured Programming (Dijkstra 76)), testing (Mills 71), operation, and maintenance stages. Some methods are specific to a particular type of problem (e.g., techniques for "structured testing" (Mills 71)), while others are much broader and more in the spirit of guidelines (e.g., ARDI (Hartman et.al. 68)).

An important outgrowth of the development of these new approaches, not yet widely recognized, is the need for software "research on research": for the development of methods, models, techniques, etc. for evaluating the successful-

ness of the various new approaches to software design and development. This research is needed because one feature most of the new methods have in common is that it is quite difficult, frequently impossible, to objectively test their efficacy (their "power to produce the desired or intended result"). More will be said about the efficacy issue momentarily.

8.1.1 Related Research.

The literature pertaining directly to evaluation of software development methods is almost nonexistent. However, some studies have been reported in related areas. Probably the most frequent target of attention has been productivity levels within the programming and system development activities. Key studies include those of Aron (Aron 70), Wolverton (Wolverton 74), Putnam (Putnam 78), and Walston and Felix (Walston and Felix 77). These studies have basically addressed two main concerns:

1. what is programmer (developer) productivity, and how ought it be measured?
2. what factors influence productivity, and how?

Walston and Felix, for example, have identified and ranked 29 different influencing factors, using as a productivity index the ratio delivered source lines of code to total effort in man-months.

Another set of studies has addressed the system development cycle. Many of these studies present normative frameworks, extracted from the authors' experience; for example, (Cooper 78), and (Cave and Salisbury 78). Others have developed models of the system development process, and use them to try to better understand the effects of various parameters on the process, and to serve as the basis for process planning and control methodologies (e.g., Putnam 78).

A few other studies address specific stages of the system development cycle - for example, software testing (Bate and Liqier 78), or system maintenance (Lientz, et. al. 77). Most of these studies also attempt to identify the important independent and dependent variables of the associated process, to develop models relating the sets of variables, and to analyze the models so as to gain further insight into the nature of the process. While none of these studies is directly applicable to the problem of determining the efficacy of a software development methodology, some do indirectly bear on the issues involved. The models developed in Sections 2 through 4 will draw on the results of these and other studies.

8.1.2 The Efficacy Problem.

Two key reasons underlying the difficulties in testing the efficacy of a new software development methodology or technique stand out. First, an objective study would generally entail the carrying out of controlled experiments in which the method under study is pitted against other potential methods (including no "method") for performing the same activity or achieving the same result. At the very least, this would require a number of parallel repetitions of a single development task, measurement of outcomes, (26) and statistical comparison of the improvement, if any, resulting from the method under study. Due to the extraordinary time and effort involved in even such a straightforward test, these kinds of studies are infeasible for all practical purposes.

Even if some extremely wealthy organization was willing to spend the money required to carry out such a test, the experiment would be seriously confounded in other ways. For instance, there would be the very difficult problem of controlling for differences between the different individuals used on the various instances of the development projects.

(26) Even defining a measurable outcome for many such tasks is often surprisingly difficult - consider the debates over the nature of program complexity (McCabe 76).

Furthermore, in many cases the situation is even more problematic. As we have noted earlier, the system development process is really a set of tasks comprising a life cycle (Figure 1.2, Chapter 1). The impact of new methods is frequently presumed to be distributed throughout all or a substantial part of the cycle. Thus experiments to test and measure a method's efficacy would have to actually develop, run, modify, and maintain numerous similar systems - thereby moving the testing task even further outside the realm of feasibility.

8.1.3 The Case of the New York Times Information Bank.

One famous case will illustrate many of the issues identified above. A fairly recent "revolution" in the programming field was initiated by Dijkstra with a now famous letter in the Communications of the ACM, entitled "GOTO's Considered Harmful" (Dijkstra 68). Out of this observation and other research by programming theorists (e.g., (Dijkstra 76), (Wirth 76)) has emerged a set of concepts and techniques termed "structured programming."

One of the chief contributors to structured programming and related new programming concepts has been I.B.M. With an eye on the commercial possibilities, I.B.M. undertook a major test of these "Improved Programming Technologies" ("IPT's"), reported in (Baker 75). The selected site was

the development of a computer-based "morgue," or information bank, for the New York Times newspaper. I.B.M.'s basic idea was to try out some of the most promising new methods in the context of a real, large system development project, while controlling those variables subject to control (e.g., project staff turnover), and making many "measurements" of the factors it deemed important (e.g., programmer productivity).

The first published reports of the success of the test were glowing. Almost incredible rates of productivity (high), error occurrence (low), and other key variables were reported (Baker 72), (Baker 75). However, on closer inspection a number of questions were raised by skeptical observers that brought the seemingly incontrovertible results distinctly into question. Some of these points were,

1. The experiment wasn't well controlled: since a number of new methods were used together on the test, it was impossible to clearly distinguish the impact of each (experimental confounding);
2. The experiment wasn't representative: many critics have pointed to the fact that I.B.M. used some of its top systems people in the project, thereby biasing the results significantly relative to what a typical organization with less skilled staff might expect;
3. The experiment wasn't repeated: results could only be compared against "similar" developments elsewhere; but there are so many variables present in such projects that the existence of "similar" projects is obviously arguable.

The debate surrounding the conclusiveness of the New York Times test, and the value of the IPT's generally, still continues today.

8.1.4 Approaches to the Efficacy Problem.

Perhaps the clearest single message provided by the I.B.M. experiment and the accompanying debate is that proving, objectively and conclusively, the efficacy of software development methodologies such as the IPT's is extremely difficult, if not impossible.

If this is indeed the case, how then can software engineers and software development researchers ever hope to know whether their efforts in new methodology development achieve worthwhile results? Strictly speaking, they can't. However, (a) to some extent it doesn't matter, and (b) to the extent that it does matter (e.g., for validating research efforts such as this one), there are some ways to partially circumvent the dilemma. The sense in which it doesn't matter has to do with how one defines the purpose of a new methodology/technique. It sometimes happens that the usefulness of a new technique will be self-evident to almost everyone (e.g., using a test-data generator as compared to generating test data by hand), and the key objective becomes one of getting people to change their behavior, to adopt the new technique. People generally are hesitant to change, even

when it is clear there is a better way to accomplish some goal. When the human cost of change is factored in, the new way may not actually be so much better after all.

In most cases, however, the fundamental net benefit to be gained from a new technique is not so intuitively obvious. (27) In these cases there are still things one can look for, test against, etc., that will serve to build a cumulative case in favor of (or possibly against) the technique in question. This is the approach taken by I.B.M. in the New York Times example cited earlier. The testing philosophy is not directed at strictly proving the case, in the objective "mathematical" sense, but rather at collecting sufficient, and sufficiently well-documented, evidence such that reasonable people would agree that the technique in question does lead to the identified net benefits. Thus whereas I.B.M.'s claim to greatly improved programming productivity using the IPT's may not have convinced the skeptics, the fact that many more firms have, to varying degrees, confirmed the benefits of the IPT's than have argued against them, lends solid credibility to I.B.M.'s arguments.

(27) There is a locally well-known quote in the handbook "How to Get Around M.I.T.": "'intuitively obvious to the most casual observer' usually means 'impossible to prove'."

8.1.5 A Predictive Viewpoint of Efficacy.

There is, unfortunately, a "deadly embrace" problem evident in the foregoing approach. On the one hand, a new technique's efficacy can only be "proven" through the collection of substantial indirect evidence; on the other hand, before deciding to use a new technique, users would usually want to see evidence of its efficacy. How does the process get started? Human inquisitiveness, the urge to try something new (and unproven), plays a role here, as does the research process (Chapter 7 being an instance thereof).

There is, however, another approach that can be taken in this regard. This approach entails taking a somewhat different viewpoint on the efficacy issue - a viewpoint we will term "predictive." From this viewpoint, we seek to identify what we expect (through introspection, judgment, limited application experience, feedback from others, etc.) to be the important effects of the methodology or technique in question. We can go further, for instance to develop a classification of the various impacts that use of the methodology is expected to produce, or even to model these impacts in a preliminary way - e.g., by suggesting possible relationships among variables, or by identifying possible ways of quantifying the potential impacts.

There are a number of benefits to doing this. First, it would provide a conceptual baseline for all parties

interested in the methodology and what it may have to offer. Second, it would help lay out for potential users of the methodology the presumed impact on outcome variables, as opposed to process variables. As an example, one view of the "chief programmer team" approach to system development (a major component of the IPT's) is that it is a new way of organizing programmers to carry out the programming task in a manner which tends to maintain high morale (a process viewpoint). Another view is that it is a way of substantially improving programmer productivity and reducing errors (outcome variables). For cost-benefit assessment of system development methodologies, outcome variables are more relevant.

Third, relationships identified in doing this might be studied further to build verifiable impact models of those aspects of the methodology, which could in turn be added to the body of cumulative evidence regarding the methodology's efficacy.

In the following sections, we take this approach and formulate some simple predictive models to delineate the classes of cost and benefit impacts that SDM would be expected to produce in a typical application. These models attempt to capture the important results - both positive and negative - that would be expected to occur as a result of adopting and using SDM in software design projects. (28)

Three major categories of beneficial impact are examined:

1. system improvements resulting from more accurate and appropriate requirements definition;
2. reduced time and cost for detailed design and implementation ("procedural design") resulting from lower communication and control overhead achievable through an improved system partitioning;
3. reduced cost for making later modifications to the final system, resulting from reduced inter-module "ripple effect."

The major category of detrimental impact addressed here is that of costs (essentially staff time) related to the carrying out of the SDM activities that would not otherwise have to be carried out.

While there are potentially many other categories of costs and benefits that might be attributable to the use of a design methodology such as SDM, it is believed that those identified above effectively capture the first-order impact. Other potential impacts are either so intangible as to be impossible to model quantitatively (e.g., the psychological impact of the methodology upon the system designers), or are assumed to be of a marginal magnitude relative to the above categories (e.g., the impact of SDM on the code debugging process).

(28) It is, to be truthful, quite tempting to over-quantify these models unjustifiably. This temptation is resisted here, with difficulty.

8.2 SYSTEM SPECIFICATION IMPACT.

A common refrain from system developers, when asked why their systems are late, over budget, or even complete failures, has been that they were unable to secure adequate user participation in the development process. In fact, a survey of over 100 implementation success factor studies by Ginsberg found "user participation" to be the single universally common factor (Ginsberg 74).

While there are many reasons why user participation is important in system development, perhaps the most important reason concerns system requirements correctness and completeness. The manifold difficulty of eliciting a user's complete set of requirements for a target system has been commented upon and studied by a number of researchers (e.g., Bell and Thayer 76). Others, such as Boehm, have argued strongly that not enough time is spent during the initial requirements elicitation and verification phase of the development process. Boehm calculates that there might be as much as a threefold return to extra effort expended during the early requirements analysis activities (Boehm 74).

One of the important impacts of SDM concerns requirements definition. SDM is "driven" by the set of system functional requirements; use of SDM forces more attention on the system's requirements than would typically be paid in a normal system development project. This comes about for three different reasons:

1. the need to develop requirements in statement form, which meet the SDM criteria (unifunctionality, implementation independence, common level of abstraction; see Chapter 3);
2. performance of interdependency analysis;
3. interpretation of a partitioning of requirements as a system architecture (see Chapter 7).

Each of these are key activities within the Systematic Design Methodology, and have been discussed earlier in this thesis. In carrying out each activity, the attention of the user and system developer are jointly focussed, in a relatively rigorous, structured way, upon the user-level (non-procedural) functional specifications of the target system. It is through this "forced," structured focussing of attention brought about by the need to carry out the steps of the methodology that missing requirements are identified, requirement inconsistencies spotted and resolved, and requirement statement errors discovered and corrected.

There is nothing magical about why this should occur. In essence, any approach that necessitates a careful, step-by-step analysis of requirements in such a structured fashion ought to achieve some of these same results. Within the SDM specifically, however, the interdependency analysis phase, and the partitioning interpretation phase, both direct extra light upon the kinds of problems that are frequently observed to occur in this work. Interdependency analysis requires the designer (and user, to a lesser

extent) to examine requirement reinforcements and tradeoffs on a pairwise basis, thereby bringing to light requirement inconsistencies and errors that might be overlooked if attention were not directed to such a specific detail level. Architectural design generation involves a general "reasonableness" assessment of the various groupings of requirements produced by the SDM graph decomposition, and experience has shown that it also helps to highlight missing requirements and requirement errors (Andreu 77d). Additional comments on the generality of this model are given later in this section.

Prior to discussing the impact model for requirements specification itself, certain underlying assumptions need to be discussed.

Assumption 1. The requirements analysis and assessment activity transpires in a series of identifiable "passes." This assumption is borne out empirically, and also through a consideration of the SDM operational mechanisms. For example, a typical SDM-oriented development effort might follow the passes shown in Table 8.1.

Assumption 2. Conceptually at least, there exists from the outset a "perfect" requirements specification - one that specifies all those requirements desired by the system's eventual users, contains no errors, is completely consistent, etc. At any point in time, this perfect set of requirements may be factored into three subsets: a set of

PASS	ACTIVITY
1	Initial problem discussion.
2	Formal expresison of initial requirements.
3	Assessment of initial requirements, and generation of revised requirements.
4	Initial interdependency analysis - identification of additional errors and inconsistencies.
5	Discussion and generation of revised requirements.
6	Interdependency analysis, decomposition, and determination of initial architecture - discovery of more errors, missing requirements, etc.
7	Final revision to requirements.

Table 8.1

Typical passes for the requirements analysis and assessment activities.

recognized requirements, a set of unrecognized but "knowable" requirements, and a set of unrecognized and "unknowable" requirements. The first set includes those requirements that have been correctly elicited. The second set includes requirements waiting to be elicited (if the designers or users could only think of them, they would recognize them as necessary), as well as corrections to previously elicited but incorrect requirements. The third set includes those requirements and corrections that at this time would

not be recognized as such even if they were brought to light.

Assumption 3. During requirements analysis step i (see Table 8.1), some proportion p_{e_i} of the remaining knowable requirements "errors" is detected. Here, "errors" should be interpreted broadly, to include incorrect statements, inconsistencies, missing requirements, etc.

The argument for proportional (as opposed to, say, linear) error discovery rate follows from empirical observation: Bell and Thayer's experiments indicate that, essentially no matter how long and hard requirements are contemplated, there will always be some remaining errors. Error decline seems to be inherently an asymptotically decreasing function of time (i.e., of number of "passes"). Also, the "satisficing" phenomenon first explored by Cyert and March (Cyert and March 64) supports this argument. They pointed out and supported the fact that people generally do not strive for the "very best" in what they do, but rather generally work hard enough at a given task to obtain "satisfactory" results, then rest awhile. In the present context, for instance, a system designer would probably not (during a given pass) seek to determine every last specification error, but rather, having unearthed a certain quantity of such errors, would relax the intensity of his analysis. A "satisfactory" result during each succeeding pass would include a smaller number of specification errors detected

than during the preceding pass, which is consistent with the assumption of some proportion of errors detected each pass.

Given the foregoing assumptions, we can sketch the two key impact relationships that make up the requirements specification model. The first of these is a declining curve of the cost of remaining errors (Curve 1, Figure 8.1). This curve is shown as asymptotic to the base level of unknowable errors, as discussed above. The other curve (Curve 2) is shown as a linearly increasing cost of staff time devoted to SDM analysis.

Curves 1 and 2 may be combined to show the net cost impact of SDM analysis on requirements errors. The combined curves indicate a minimum point, t^* , in Figure 8.1, which represents the breakeven point for time spent on SDM analysis, as far as reducing requirements errors goes. It should be noted that specification error reduction is the only payoff factor being considered in these models that is significantly time dependent. Put differently, assuming SDM is used at all, the payoffs to be discussed in the next two sections will, in theory, occur. Only the payoff due to specification quality improvements depends, to an important extent, on how much time is spent on the requirements analysis activity.

Another important point is the question of whether this error-reduction effect would occur with any method of

repeated perusal of the requirement statements. While some error reduction would almost certainly occur, there are good reasons to believe that SDM analysis is especially well-directed toward achieving these ends. Our experience has shown that many of the mental clues that lead to improvements and discovery of errors in the requirement statements stem from having to think about the requirements

a. repeatedly,

b. in a step-by-step, structured way,

c. from different viewpoints,

as is required in SDM analysis. Initially, the requirement statements are considered as requirement statements, looking for ambiguities, mis-statements, etc. Later they are considered during interdependency analysis in the light of the designer's mental models of implementation. This tends to shed a whole new light on each statement, thereby turning up different errors, problems, etc. Finally, they are considered during system structure interpretation, together with other, closely related requirements, turning up still other types of problems.

Most individuals rapidly develop mental blocks, "blind spots," etc. when going over the same statements time after time from the same viewpoint, with the same objectives in

mind. By having the designer focus on alternative viewpoints and objectives for each SDM "pass," these mental blocks are to a considerable degree mitigated.

Another interesting point that this first impact model makes clear is that it is theoretically possible to spend too much time on analysis activities, i.e., carrying on past the point t^* in Figure 8.1. This, however, is unlikely. In general, the problem with requirements analysis has been that, due to its unstructured nature and due to lack of user participation mentioned above, far too little time is usually spent on it in the course of a typical systems development project. In such cases the model's operating point would lie well to the left of the optimal point t^* . Any "reasonable" amount of effort directed toward SDM analysis would most likely not move the operating point as far rightward as t^* .

* * * * *

In the next section we consider the nature of the impact SDM should have on a different aspect of system development: the costs of communication and coordination among individuals and teams in a major development project.

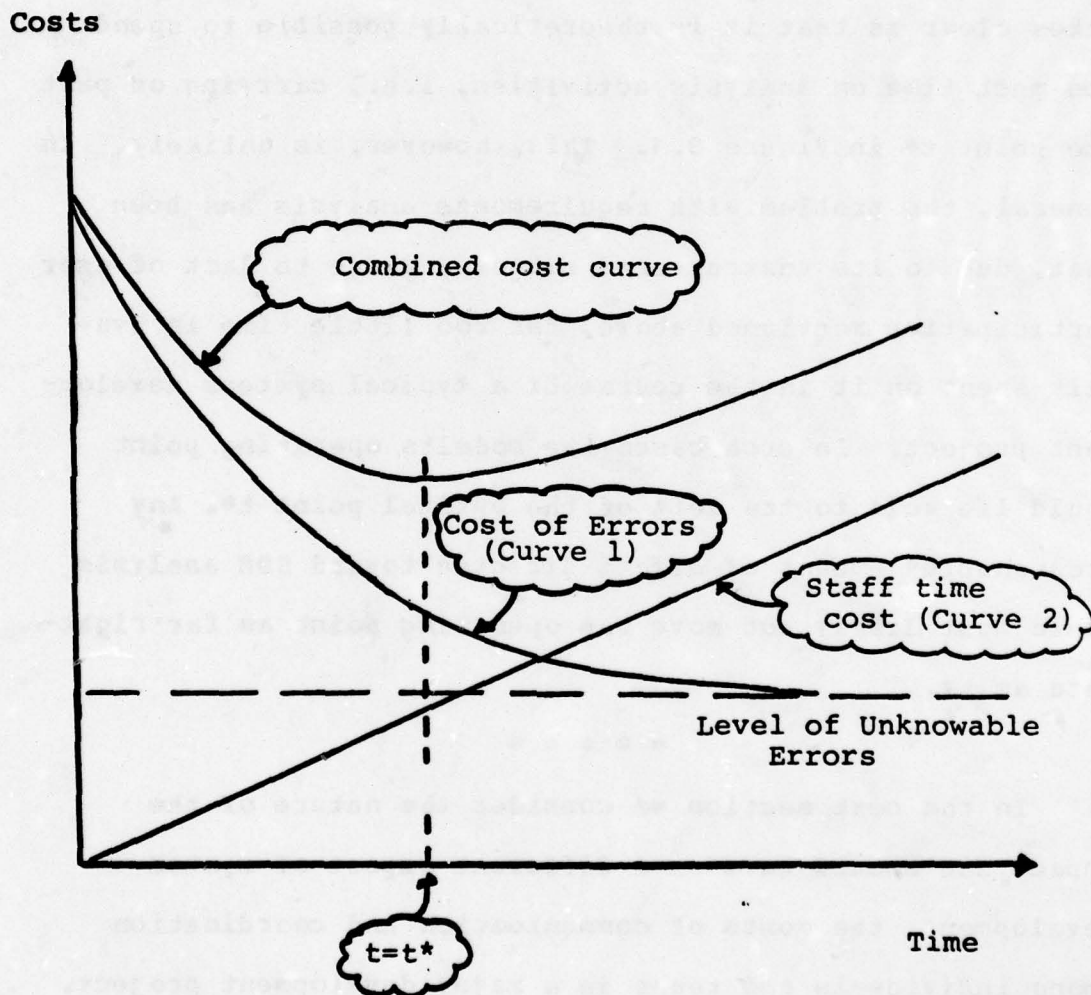


Figure 8.1

Graph sketches of the relationships for the requirements specification impact model

8.3 SYSTEM PROCEDURAL DEVELOPMENT IMPACT.

The second major area of SDM impact concerns the time required to carry out the procedural development (detailed design, programming, and testing) phase of the system life cycle. As a number of authors (e.g., Brooks 75, Scott and Simmons 75) have pointed out, a major impediment to these activities is the need for substantial coordination and communication among the various individuals and groups of people involved. Indeed, Brooks has cited this need as the reason why "men and months are not interchangeable" in system procedural development activities.

There are two primary costs that arise as the need for coordination and communication grows:

1. the cost of additional time that the development staff must spend in these activities, which detracts from productive development;
2. the cost of additional layer(s) of project management overhead needed to organize and manage the inter-group coordination and flows of communication.

An example of (1) would be the time spent in intra-group meetings devoted to ironing out issues and misunderstandings that arise from interdependencies between the various tasks. An example of (2) would be the resources consumed in that portion of the formal project management process devoted to managing intra-group issues (e.g., designing a properly sequenced module testing plan).

Parnas (Parnas 75) has pointed out the importance of partitioning the procedural development effort so as to minimize the required interactions among designers, programmers, etc. SDM provides, in the system functional architecture that emerges from decomposition analysis and interpretation (see Figure 2.1 and Figure 7.2 in earlier chapters) a blueprint for subdividing the procedural development tasks in a way that effectively meets Parnas's goal. Each subproblem within the architectural design should, if possible, be made a separate focus of procedural development effort. In some cases it may be appropriate to combine together two or more subproblems to form a larger subproblem for these purposes (e.g., for distribution to a limited number of different detailed design groups), although splitting subproblems for this purpose should obviously be avoided. If this procedure is followed, the functional requirements will be partitioned for procedural development in a way that should keep to a minimum the needs for inter-group coordination and communication.

We can generalize the notion of a good requirements partition in order to develop a model of the communication/coordination cost impact. We define a variable D as the density of module interconnectedness. Clearly, the impact of D upon the two factors identified is direct: that is, as D increases, each of these factors will increase

also. The graph in Figure 8.2 shows a hypothetical set of curves that depict these relationships.

The first function (Curve 1), relating interconnection density to additional "non-productive" staff time, is illustrated as a nonlinear function that "blows up" at a certain high level of density. As interconnection density D increases, a larger and larger proportion of staff time must be devoted to the coordination and communication functions, thereby shrinking the time available for productive development (given some initial timetable) toward zero. Conceptually at least, there exists some (high) level of problem complexity, embodied as a large D value, such that the coordination and communication needs among system development staff use up essentially all available time.

Evidence for this phenomenon is cited by Haney in the context of development and maintenance of a Honeywell operating system (Haney 76), and by Belady and Lehman, in the context of the IBM OS/360 operating system (Belady and Lehman 76).

There is an interesting parallel between the asymptotically growing need for coordination and communication discussed above, and the phenomenon of "thrashing" in a demand-paged virtual memory system. In the latter case, a "high D value" would correspond to a lower level of reference locality for a given set of active processes. A pro-

cess with a low level of reference locality is essentially one in which the various sub-parts are "tightly interconnected" (in terms of execution references over time), much like a tightly interconnected set of modules in a system architecture that exhibits high D. As locality decreases (D increases), the system overhead costs rise in a distinctly nonlinear, accelerating fashion, similar to that suggested by Figure 8.2.

In contrast to the first function, the relationship between extra development management overhead and D is hypothesized to be one of linear growth (Curve 2). The evidence for this comes indirectly, from researchers such as Wolverton (Wolverton 78), who suggests that such overhead grows linearly with project size. While D does not necessarily purport to measure size directly, it is reasonable to argue at a first approximation that a doubling of interconnectedness complexity should demand more or less proportionally equivalent management response to that demanded from the size doubling. For example, it requires some, but not a disproportionate amount, of additional project management time to convene a meeting for four team heads as between two, or to send design change documentation to four teams as compared to two.

It is not suggested that these functions are fully representative for all values of D. In particular, D values

outside the range $[0, D^*]$ are clearly inapplicable in this model. But even within this range, there exists a smaller "relevant range" within which the functions - especially the first - are really hypothesized to hold. This restricted relevant range is typified in Figure 8.2 with dashed lines.

We have formulated the coordination/communication impact model in terms of the hypothetical variable D . It is important to note that a very reasonable practical surrogate for D would be the measure of inter-module coupling, C , used in the SDM decomposition analysis. Furthermore, a goal of SDM is to determine a system structure with low C , which in turn implies low D . That is, other things being equal, use of SDM should shift the operating point toward the lower left along the combined curve of Figure 8.2.

* * * * *

A third class of impact of SDM on project development is discussed in the next section. There we propose a model for the effect that a system's design which is based on a high-strength, low-coupling early partitioning of requirements has upon costs of maintenance and modification later in the development cycle.

Staff time for
Coordination/communication
Overhead

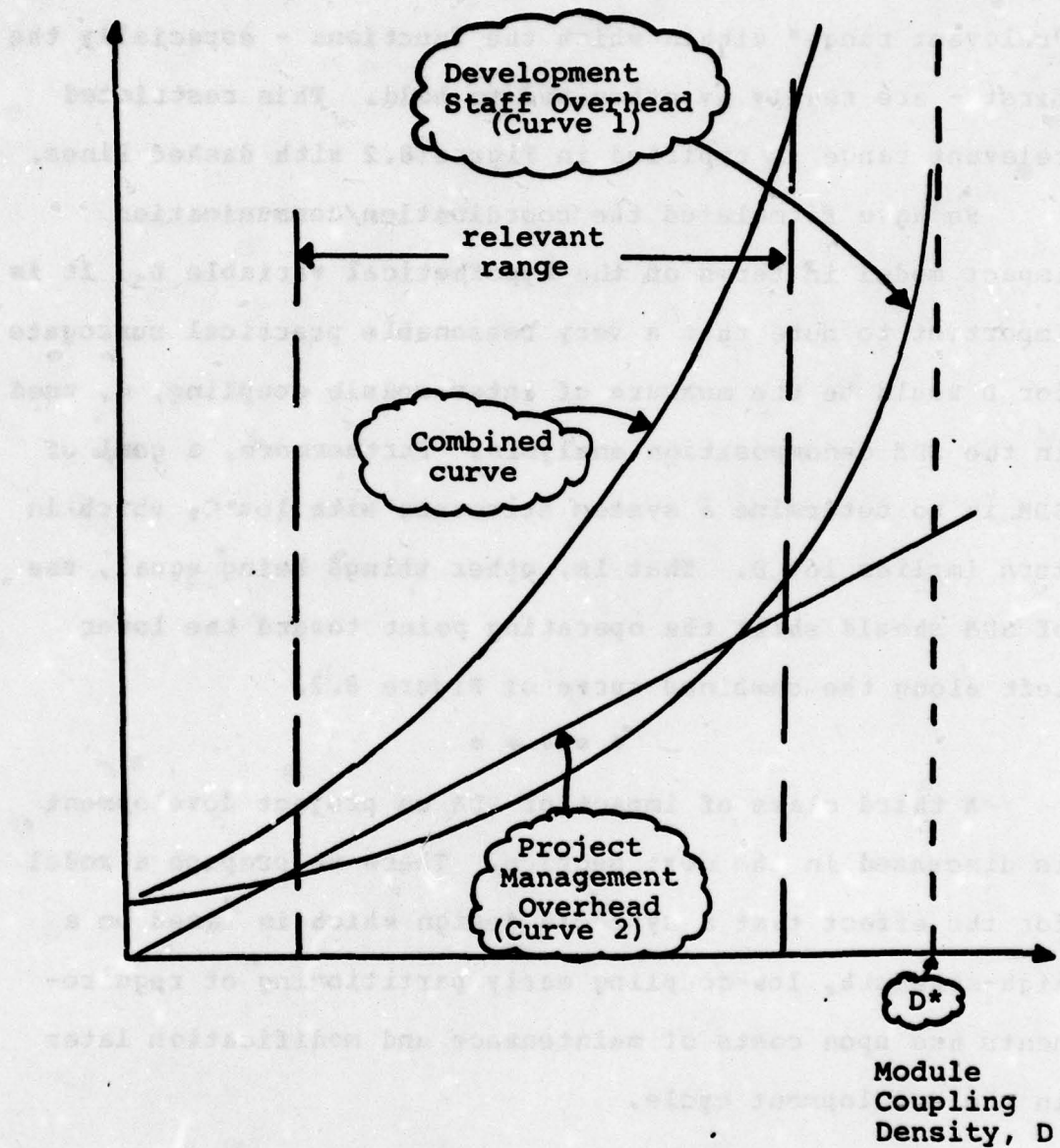


Figure 8.2

Graph sketches of the relationships for the
communication/coordination impact model

8.4 SYSTEM MAINTENANCE/MODIFICATION IMPACT.

The third major potential area of impact for the Systematic Design Methodology concerns the cost of maintaining - and, especially, modifying - large software systems. The rapidly inflating costs of system maintenance/modification have been commented upon by numerous authors (e.g., (Boehm 75), (Yau 78)). The need to devise ways of designing and constructing systems so as to reduce the maintenance load and ease the costs of later modification has been widely recognized. One study, for example, has estimated the production cost of a software product to be about \$75 per line of code (LOC), while the maintenance cost per LOC of the same system was estimated at \$4000 over the system lifetime (Boehm 75). A variety of studies have indicated that anywhere from 40 to 80 percent of the original system development costs are eventually spent on "simple" maintenance for the system; when all post-implementation work (including non-trivial modifications) is taken into account the figure rises to 200 to 400 percent (Thayer 77; Goetz 78). At any rate, while often exhibiting rather wide variances, these studies unambiguously indicate that software maintenance and modification functions are assuming increasingly high profiles, and that good system design practice must take this fact fully into account.

It is common practice to differentiate between "maintenance" and "modification" of software systems. The former term refers to the fairly large number of relatively minor changes, bug fixes and improvements that may be made to a software product following its initial release. Examples include a specific patch to fix a minor error, or the addition of an extra report to a batch-oriented DP system.

In contrast, software modification generally refers to more significant changes made to the system - changes that usually entail some amount of redesign, and that impact most or all of the system's users. Major changes are usually undertaken to add significant new functions to the system, or to improve the operation of a major portion of the system. A prime example is a new release of a vendor's operating system (e.g., IBM's OS/360).

For the purpose of assessing SDM's impact, we will focus primarily on the modification function. More precisely, we will be concerned with those changes to the user-visible functions provided by a particular system - changes which are of significant enough scope that impacts on multiple system modules, or major components, are likely. While arguments could be made that SDM would impact both maintenance and modification costs, the latter impact is almost certainly the more significant.

The primary cause of system modification is the recognition on the part of the user clientele for changes to the functions provided by the system, including such things as addition of completely new functions, major enhancements to present functions, important improvements in efficiency (response time, turnaround time, etc.), availability, reliability, etc., and changes to fix major system problems. One primary mechanism through which system modifications exert their apparently disproportionately high economic impact is the rippling effect that such changes have on the system. Changes to one system component very often result in the need to make subsidiary changes to other modules. The propagation of these indirect changes results in telescoping of the effect of the original change: a single change can cause other changes, which can in turn cause still other changes, etc. When viewed in this way, the possibility of an instability phenomenon - a single change generating a never-ending sequence of subsidiary changes - presents itself. In fact, evidence exists that such an unstable situation could occur, and may have been closely approached in certain real-world systems (Haney 76; Belady and Lehman 76).

Since the changes to be implemented are not known a priori, it is not possible to model the change propagation effect deterministically. However, some simple probabilistic arguments provide considerable insight. Consider two system modules, M_i and M_j . Using the SDM framework, these

modules are "linked" to the extent that the requirements represented within one module are interdependent with the requirements represented within the other. The greater the number of such interdependency links relative to the size of the module, the greater the likelihood that a change to one or more of the requirements in the first module will impact (cause a change in) the other.

Suppose we let p_{ij} be the probability that a change to module M_i causes a change in module M_j . (29) The probability p_{ij} would be directly related to the strength of the interconnection between M_i and M_j (see Section 7.4 for an analysis of the interconnections strength in the Budgeting System case study). Also, p_{ij} would be inversely related to the size of (number of requirements in) module M_i , since p_{ij} is (conceptually at least) an a priori value. That is to say, we don't know ahead of time which requirements it will be necessary to change later on; therefore, we can only say that, for a given interconnection strength, the larger the number of requirements in module M_i , the smaller the likelihood that a change will impact a requirement interdependent with other requirements in module M_j , hence the smaller

(29) Note that we are concerning ourselves here with changes to original functional requirements, and the rippling effects related thereto. Other types of modifications, for example the addition of totally new functions, should be viewed in terms of the changes they bring to bear on previously implemented functions.

should be p_{ij} . As a consequence of this, $p_{ij} \neq p_{ji}$ in general.

The above reasoning leads us naturally to define a "change propagation likelihood matrix," P :

$$P = [p_{ij}] .$$

The (i, j) th entry in P is the probability that a change to module i necessitates a further (new) change to module j . The change propagation likelihood matrix can now be used to study the cost impact of system modifications. Suppose at some point in time a set of changes to the system is being considered. Let

a_i = the number of changes to
module i being considered.

The term a_i may be thought of as the number of known "bugs" in module i at a point in time (where "bug" is to be broadly interpreted - e.g., an incompletely implemented function demanded by system users would be an example of a "bug"). Let

$$A = [a_1, a_2, \dots, a_n] .$$

A is the vector of planned module changes; A will be termed a "revision" to the system.

Now, the originally planned changes a_1, a_2, \dots, a_n give rise to additional changes. If, for instance, there are a_j planned modifications to module j, this will generate $a_j p_{j1}$ expected number of changes to module 1, $a_j p_{j2}$ changes to module 2, and so forth. The overall expected impact of the planned changes - the average number of "second-level" changes - is simply the matrix product AP. The first element of the row vector AP is the expected number of second-level changes to module 1, etc.

The second-level changes themselves give rise to yet additional changes, in a corresponding manner. The expected number of third-level changes to each module can be calculated as

$$(AP)P = AP^2.$$

Conceptually, this telescoping of module changes continues ad infinitum. Mathematically, the total expected number of changes to each module may be calculated by summing the resulting infinite series,

$$\begin{aligned}
 T &= A + AP + AP^2 + AP^3 + \dots \\
 &= A(I + P + P^2 + \dots)
 \end{aligned}$$

If all the eigenvalues of the matrix P are real and lie in the range $(-1,1)$, then a basic result in matrix algebra (Strang 77) says that this matrix series converges, to the value

$$T = A(I - P)^{-1}.$$

Essentially, the expression above summarizes the ripple effects that occur as a result of modifications to modules of a large system.

Some insight can be gained into the nature of this model by considering some special cases. Suppose the change propagation likelihood value p_{ij} is

$$p_{ij} = \begin{cases} p & \text{if } i = j, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Then a single modification made to any one module results in a total number of expected changes to that module of

$$1 + p + p^2 + \dots = 1/(1 - p)$$

For instance, if there is a 10% chance of a change in module i resulting in another change in the same module, then $p = 0.10$, and the expected number of changes resulting from a single change to module i would be $1.1111\dots$.

Now assume that there are n modules, and $p_{ij} = p$ for all i, j . That is, the likelihood of change propagation between any pair of modules is $100p\%$. Then a single original change to one module generates np expected second-order changes, each of which generates another np expected third-order changes, etc. The total number of changes is then

$$1 + np + (np)^2 + (np)^3 + \dots = 1/(1 - np) .$$

This expression reflects the impact of both change propagation likelihood p and system size, in terms of number of modules, n .

There is an asymptotic limit at the point $np = 1$, or $p = 1/n$. This limit represents the point at which the system is large enough and interconnected enough that a single change will ripple forever throughout the system - the number of generated changes "blows up." This blow-up phenomenon is closely related to the point $D = D$ discussed in the Procedural Development Model of Section 8.3.

This basic module connectivity model gives us insight into the impact that SDM ought to have on system mainte-

nance/modification. SDM guides the system architect in devising a high-strength, low-coupling structure which may be used as a framework for constructing the detailed modules of the target system. The fundamental assumption underlying SDM is that the methodology leads the system architect to a better (higher-S, lower-C) structure than would otherwise evolve. While the veracity of this assertion is testable to some degree, such tests have only been carried out in subjective fashion (e.g., by asking the designers' opinions - see Chapter 7). Our best indications to date are that SDM indeed leads to a better system structure than that which a typical designer, using only his judgment and past experience, is able to determine.

This may be translated into the terms of our maintenance/modification model by saying that we would expect the P-matrix entries for an SDM-based structure to be smaller than those for a non-SDM-based structure. This is equivalent to saying that the ripple effect for a high-strength/low-coupling (i.e., high M) system is lower than for another similar system with lower M.

The potential cost impact of such an improvement in system structure has been illustrated by Haney (Haney 75). He pointed out that informal experiments with the Xerox Universal Timesharing System showed each change to be causing approximately 10 additional changes. This system had about 22 major modules, so that

$$1/(1 - np) = 1/(1 - 22p) = 10 + 1 = 11 .$$

Hence $p = 0.04$. If the interconnection propagation likelihood could have been reduced by 25%, to $p = 0.03$ (say, through the use of SDM in the early system architectural design phase), the number of additional changes following a single change would have dropped from 11 to 3! This would obviously lead to substantial savings in maintenance costs.

8.5 SUMMARY.

In the previous three sections, three different models, one for each of the major impact areas of the Systematic Design Methodology, were described. To recap briefly, the three areas are:

1. system specification - correctly identifying and stating as many of the "knowable" functional requirements as possible, within the constraints of time and manpower available;
2. system procedural development - carrying out the detailed design, coding, and testing of the system, together with the accompanying load of coordination and communication overhead among members of the development team(s);
3. system maintenance/modification - making necessary or requested alterations to the functions provided by the system - correcting errors, adding new functions, etc. - while simultaneously making sure that all secondary, tertiary, etc. changes to other system modules are also identified and carried out.

The three models developed in the previous sections are different from each other in certain ways. For one thing, the first two are deterministic, while the third contains probabilistic elements. Also, in the first model, time is the key independent variable, whereas in the second the focus is on the density of module interconnectedness, and in the third, the number of changes to be made to different system modules together with the probabilities of change propagation. All three models attempt to capture the effects of the modelled independent variables and parameters

on cost - i.e., various components of cost are the dependent variables in the different models.

The models and concepts presented and discussed here are in many ways crude and of limited scope. But, as we pointed out earlier, we were able to find no other published work on modelling the impacts of design methodologies at all. This preliminary effort at least illustrates the feasibility for improving understanding of the economic potential of improved functional design partitionings and other related aspects of systems design.

A number of issues were raised but not completely answered in this report. Some of the most important are:

1. how to accurately determine the various parameters of such models;
2. how to better verify the correctness of the models' functional forms;
3. how to make effective use of such models - e.g., what can be learned from sensitivity analysis or parametric variation?
4. how to improve the models - e.g., perhaps we ought to be looking at different aspects; perhaps some of the "second-order" effects are really more important than they were assumed to be here.

These and other questions are very appropriate issues for further SDM research, and are examined further in the final chapter of this thesis.

Chapter IX

CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH.

At the outset of this work, a conscious decision was made to take a "broad brush" approach in trying to advance the Systematic Design Methodology on a number of fronts. While perhaps less common for doctoral thesis research, this approach was felt to be necessary because the problem area being researched (software architectural design) is a new and largely undeveloped area. The seven major fronts addressed in this thesis were:

1. Basic philosophy and concepts;
2. The SDM modelling framework;
3. SDM decomposition analysis techniques;
4. The linkage of the SDM to preceding and following activities within the system development cycle;
5. Efficacy of the SDM;
6. Development of the computer-based analysis package for application of the SDM;
7. Testing the SDM.

* * * * *

In considering the accomplishments of this thesis in summary, it must be kept in mind that the thrust of the work has been methodology building - not, for example, hypothesis testing or theorem proving. Thus the relevant questions are, how far and how well was the methodology developed? Most of the comments in this chapter are directed towards those questions.

This thesis largely built upon the foundations laid earlier by Andreu (Andreu 78). These foundations proved solid, and continue to endure. While many of the earlier approaches have been advanced or replaced by the results described herein, very few of Andreu's contributions were found to be incorrect in any essential respect. A number of his ideas continue to be applied unchanged.

In retrospect, it is safe to say that there is essentially no research literature, Andreu excepted, addressing this problem area directly. This may seem surprising, seeing that software engineering as a discipline is well into adolescence now.(30) It would seem that software "engineers" do not view software architectural design as within their sphere; or if they do, they have not yet come to grips with it as a researchable issue. SDM hopefully will begin to change this state of affairs.

(30) Its origins stem roughly from the 1968 NATO Conference (Randell 69).

There is a tendency in much of the software requirements-related literature toward hype - a kind of "our methodology can do it all" argument line. (Hopefully we have not done that here.) A good part of this is the result of terminology confusion and variability. In reading many of the papers describing current thinking, one often wonders how large is the gap between what an author claims his approach is capable of accomplishing, and what it has actually accomplished. For instance, one paper discusses (at a theoretical level) the possibility of capturing and modelling the functional requirements for a software system, then performing simulations to test the operational soundness of the system. Talking about simulating the detailed operation of a system described solely in functional terms basically suggests doing away with the procedural development phase of the development cycle altogether! The requirements concepts described in Chapter 3 oriented certain key aspects of system functional requirements into a common framework, and attempted to bring them into better focus.

Investigations into the decomposition analytics (mainly Chapters 5 and 6) are the most clearly structured part of this research. In particular, the interchange partitioning algorithm stands as one of the major contributions of this work. There is a tendency, however, to spend an undue amount of time on these aspects of the problem. As far as

SDM is concerned, this may lead to a certain illogic - not unlike calculating a function to five decimal places when the input data is only accurate to two. This is counterbalanced by the fact that the decomposition algorithms might well have applicability beyond SDM itself. The strength-coupling decomposition objective function is quite general in nature, as are many of the other analysis techniques (e.g., use of weighted, undirected graphs). The graph/network/clustering literature was studied mainly to seek out ways of attacking the SDM decomposition problem. Now it would be appropriate to look there again, to see whether what we have learned regarding SDM analytical methods has applicability to other classes of problems. cursory study to date indicates a number of possibilities - e.g., database organization so as to minimize paging interrupts (discussed further in the next section). Another possible application, totally unrelated to SDM, came to light recently. It concerns spheres of corporate influence. Emerging from the phenomenon of interlocking corporate directorships, this problem involves how to isolate groups of corporations that are especially tightly "coupled," to determine which corporations fall in which groups, whether such groups exert noticable special influences on the economy, on smaller corporations, etc. One technique that has been used to model this problem is the weighted, undirected

graph (Levine 72) - the same model used in SDM. An initial examination suggests that the SDM analytical tools may be directly appropriate for determining spheres of influence also.

Despite the earlier warning, there are a number of intriguing issues for further study as regards the analysis techniques. For instance, it was noted earlier, in decomposing the Budgeting System requirements graph (Chapter 7), that a fairly wide range of "best" decompositions occurred using the different techniques. HIER1 located a best decomposition with an M of 0.05, HIER3 with 0.67, and INTERCHANGE with 0.85. This leads us to ask whether there might be even better decomposition methods waiting to be developed and brought to bear on this problem. More on this in the next section also.

One of the important contribution of this thesis was testing the use of SDM in a real design context. Prior studies had been carried out by the person doing the SDM research, hence were somewhat unrepresentative of the true reactions of real system designers. Our application to the M.I.T. Budgeting System provided a number of useful lessons and insights, while basically confirming the soundness of the approach and the perceived usefulness of the results. Over the course of ten meetings, the system architects for the new Budgeting System (two key individuals, and a third less centrally involved) and this investigator worked out

the functional requirement statements and requirement interdependencies in detail. The decomposition of the resulting requirements graph, and the interpretation of the design problem structure, was then carried out, as described in Chapter 7. The final architecture for the target system was well received by the Budgeting System architects, and they have expressed their intention to use this architecture in guiding their coming detailed design work. Exactly how this ought to be done - the subject of linkage of architectural design to detailed design - is an important area for further research, and is discussed again in the next section.

The SDM application to the Budgeting System was quite successful, as far as it went. SDM "worked" in a technical sense, and it was definitely perceived to be useful and efficacious by the (presumably unbiased) system designers. However, how are we to know whether it really will lead to a better system design? This is the "efficacy problem" addressed in Chapter 8. It is a difficult issue, and Chapter 8 only served to scratch its surface. In that chapter we presented three models of the impact we believe SDM would have on the system development process and economics. We basically took a cost-benefit view of the efficacy question, which is only one view. We argued there that the problem of literally proving efficacy is extremely difficult, although a "cumulative evidence" approach could be used as a surro-

gate to literal proof. Regardless, the question of whether SDM really does what it is supposed to do - i.e., lead to better system design - is clearly important, and is yet another area for additional research. This and other potentially fruitful research directions are discussed in more detail in the next section.

9.1 DIRECTIONS FOR FURTHER RESEARCH.

Since, as we have already said, the software architectural design field has been explored very little, there is a wide variety of potentially fruitful directions for further research. To limit the scope of this brief discussion, we will address only those directions related to SDM, and those which we believe to be the most important and promising.

Linkage.

Probably the single most important problem to be addressed in future SDM-related research concerns the question of linkage of an SDM-based design problem structure to the system detailed design process. As was pointed out earlier (Chapter 8), this linkage involves a transition from the functional development to the procedural development phase of the system development cycle. Crossing this threshold entails major changes: in the nature of the design/development work, in the primary modes of thought (solution-oriented rather than problem-oriented), etc. It would be very interesting to investigate formalisms for converting an SDM-based architectural design into a detailed design blueprint. It seems quite unlikely at this point that "automatic" means for performing such a conversion could be devised. However, informal, perhaps diagrammatic

approaches might be devised. Requirement specification mechanisms such as PSL (Chapter 3), or DeWolf's concepts (DeWolf 77), might be brought to bear. Also, the information contained in the interdependency descriptions represent a tie from the functional to the procedural phase and might be used to advantage here.

One approach that ought to be studied is hierarchical structuring. Recall that in developing functional requirement statements for SDM, one of the guiding principles is to frame all the statements at the same level of abstraction. This is the main reason why hierarchical implication relationships, proposed in Chapter 4, turn out to not buy the designer very much: they are largely factored out as a result of the common abstraction level principle.

However, it is well known (e.g., see (Simon 69)) that hierarchical structuring is a very common and powerful way for humans to deal with complex systems. Therefore it makes sense that there might be some effective way of utilizing the power of hierarchical structuring principles in building a bridge from functional development to procedural development. For instance, perhaps each design subproblem could be used to generate a set of procedural detailed requirement specifications, which could then be modelled using, possibly, a representational mechanism such as that proposed by DeWolf. There are surely many other untapped ways to make use of the hierarchical structuring ideas also.

At the present stage of SDM's development, it is expected that an SDM-produced architecture would be used by software designers in a largely judgmental fashion (e.g., by basing the partitioning of the detailed design effort on it). It would, therefore, be useful to study more deeply how this could, or ought to, take place. Of course, the best way to learn more about this process is to actually do it (experimentally), but that approach runs rapidly up against the problem of feasibility for a small research effort. A second approach would be to "talk about it" with system designers, as was done briefly with the ESD Budgeting System designers (Chapter 7). A third, perhaps most promising, approach would be to track a real detailed design effort for which an SDM architecture had been earlier developed, to determine as precisely as possible how and why SDM influenced the follow-on design effort and outcomes. This sort of tracking study could be carried out with the Budgeting System in the future, assuming the BSD designers are willing to participate.

Study of Non-methodological Architectural Design Process.

Another potential avenue for further research would be to examine carefully how real system designers actually effect a design problem structure. We have suggested, primarily on the basis of limited direct experience, that this

AD-A074 911

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC METHODOLOGY FOR DESIGNING THE ARCHITECTURE OF COMP--ETC(U)
SEP 79 S L HUFF, S E MADNICK
CISR-P010-7906-12

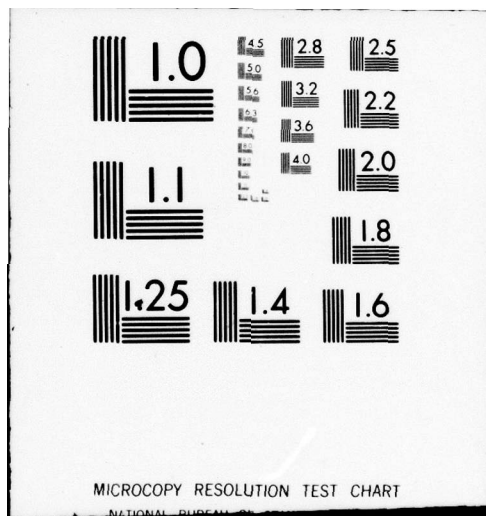
N00039-78-G-0160

NL

UNCLASSIFIED

6 OF 7
ADA
074911





task is dealt with intuitively, partly subconsciously, based on prior experience with similar tasks. Buried in this process somewhere may be valuable clues for ways to improve SDM. Of course, it is not necessarily correct to try to emulate a designer's thought process directly in this regard. After all, one of the major assumptions underlying this research (an assumption frequently verified by example) is that the judgment-based approach is often not the best. Nevertheless, it would be worthwhile to study how a designer does (to the extent that he even does) carry out this task, with the hope that it would lend more insight into the effectiveness of the SDM approach.

Efficacy.

The efficacy question was explored in some detail in the previous chapter. It will not, however, go away that easily, and it continues to present a very challenging problem. How are we to be sure that SDM leads to a good system design, or even to a better design than would be obtained without SDM? Is SDM something truly worthwhile for system designers to use, and how can we justify the answer? To date the bulk of our SDM research has been focussed on building the methodology. On this dimension we have come a reasonably long way. If the methodology is ever to be "sold" to the outside world, more work will have to be done on the efficacy issue.

Analytics.

The important analytical techniques for applying the SDM concepts are reasonably well developed at this point. It may well be argued that other issues - linkage, efficacy, etc. - deserve more future attention than do the analytical methods. However, the latter are probably more tractable, better structured problems. For instance, it was pointed out in Chapter 5 that there exist certain graph decomposition techniques that have not been tried in the SDM context. McCormack's approach (see Chapter 5, Section 5.3.4.2) appears especially promising. It is likely that additional study of the clustering and network literature could unearth still more potentially useful approaches to the decomposition and requirements modelling problems.

There is an inverse research issue to be considered here also. It is quite possible that some of the analytical methods developed in the SDM work might be useful in other contexts. To name one possible area, researchers have tried applying simple graph decomposition techniques to the computer database organization problem: to organize files so as to keep close together those data items that tend to be referenced together; this involves concepts such as spacial and temporal locality of reference (McCabe 77), (Morgan 76). The tools developed for SDM decomposition ought to be tested on these related problems also.

Application Studies.

Another important avenue for further research involves additional application studies, preferably with real system designers working on a real development project such as the case study described in Chapter 7. At this time, one additional such study is underway, involving the design of a computer-based electronics testing system. This study has the potential of shedding new light on the applicability of SDM to designing systems consisting of more than pure software, i.e., hardware-software systems. Other application studies ought to similarly be directed toward untried design contexts. (31) A promising area of hardware-software applications is real-time systems, wherein much of the detailed design methodology development work to date has been carried out (Davis & Vick 77).

Another aspect of future application studies concerns techniques for using the methodology. In the Budgeting System study, both individual, individual followed by group, and group-oriented analysis methods were tried, although no attempt was made to monitor the relative effectiveness of the different approaches. It is possible that for a larger application, a Delphi-like technique could be used to advance

(31) Holden (Holden 78), in deciding what would be the target of his SDM application study, debated between the design of an operating system (his eventual target), and the design of a submarine!

tage. Here, different system architects would work through portions of the SDM analysis, then, while studying the same analysis work of other team members, would be allowed to modify his own assessments based on the feedback from the other analysts. This approach would likely lead to a convergence of design opinion (e.g., regarding the existence or strengths of interdependencies) less susceptible to the bias of an open meeting approach such as that used in the Budgeting System study.

Also, there are interesting questions such as whether certain personality types are more adept at SDM analysis than others. Finally, ways to streamline or improve the effectiveness of the analysis activities could be studied. For instance, in doing the Budgeting System interdependency analysis, we debated the potential usefulness of a dual projector scheme for being able to easily display any pair of requirements side-by-side. It was felt that this would make the analysis activity much simpler, by avoiding having to constantly flip pages of paper back and forth.

Analysis Package.

A serious effort was made in the development of the current version of the PL/1-based SDM analysis package to make it easy to use, understand, and hence modify. In particular, it should prove especially easy to incorporate new

decomposition algorithms into the MASTER system, either for experimental or "production" purposes. Many of the subprograms currently making up the package are general-purpose, and would be useable by new routines (e.g., the subprograms to calculate the decomposition objective function M for any graph).

One consideration for further work on the analysis package could involve incorporation of some of the stand-alone routines developed during the Budgeting System project to manage text-oriented files. While these routines have proven very useful, they are not as yet built into the MASTER package (see Appendix D), although doing so should not be difficult.

In making future changes to the MASTER analysis package, the documentation in Appendix D, together with the program listings, should provide all the required background information.

* * * * *

There are undoubtedly many other avenues for further study in the Systematic Design Methodology and related research areas. While a thesis document necessarily avoids exuberant phraseology, it is fair to say that working in an unexplored yet clearly important research area such as this

one is both exciting and difficult. The excitement comes from the feeling that one is helping to open up and contributing to a whole new research problem area. The difficulties include the lack of much of a base to build upon, the lack of a contingent of co-researchers to interact with, and the occasional fear that perhaps the whole thing really is not of much value. Having struggled with the concepts and methods described here (simplistic as they may be) for over a year's time, I can fairly say that I, for one, am convinced of their potential for improving the quality of the designs for complex systems. I hope the reader is convinced also.

REFERENCES

- Aho, A., et. al.: The Description and Analysis of Computer Algorithms, Addison-Wesley, 1974.
- Alexander, C.: Notes on the Synthesis of Form, Harvard University Press, 1964.
- Alford, M.: "A Requirements Engineering Methodology for Real-time Processing Requirements", IEEE Trans. Soft. Eng., vol. 3, no. 1, 1977.
- Altman, J., et. al.: Handbook of Methods for Information Systems Analysts and Designers, NTIS Publications, AD725782, 1971.
- Anderberg, M.: Cluster Analysis for Applications, Academic Press, 1973.
- Andreu, F. C.: "Set Decomposition: Cluster Analysis and Graph Decomposition Techniques", Internal Report P010-01-01, M.I.T. Sloan School, September 1977(a).
- Andreu, F. C.: "Solving Decomposition Problems: Alternative Techniques and Description of Supporting Tools", Internal Report P010-01-02, M.I.T. Sloan School, September 1977(b).
- Andreu, F.C., and S. E. Madnick: "An Exercise in Software Architectural Design: From Requirements to Design Problem Structure", Internal Report P010-01-05, November 1977(c).
- Andreu, F.C., and S. E. Madnick: "Completing the Requirements Set as a Means Towards Better Design Frameworks: A Follow-up Exercise in Architectural Design", Internal Report P010-01-06, MIT Sloan School of Management, December 1977(d).
- Andreu, F. C.: "A Systematic Approach to the Design and Structuring of Complex Software Systems," Ph.D. Thesis, Sloan School of Management, M.I.T., Cambridge, Mass. 1978.

- Archer, L. C.: "Systematic Method for Designers", Parts I-VI, Design, No. 172-188, April 1963-August 1964.
- Aron, J.: "Estimating Resources for Large Programming Systems", in Buxton, J., and J. Randell (eds.): Software Engineering Techniques, NATO Science Committee, Brussels, 1970.
- Aron, J.: The Program Development Process - Part II: The Development Team, preliminary draft version, Addison-Wesley Publishers, June 1977.
- Baker, F.: "Chief Programmer Team Management of Production Programming", IBM Systems Journal, vol. 11, no. 1, Jan. 1972.
- Baker, P. T.: "Structured Programming in a Production Programming Environment," IEEE Trans. on Soft. Eng., vol. 1, no. 2, June 1975.
- Bate, R., and G. Ligler: "An Approach to Software Testing: Methodology and Tools," Proc. Third Conference on Software Engineering, IEEE Publications, 1978.
- Belady, L., and M. Lehman: "A Model of Large Program Development," IBM Systems Journal, vol. 15, no. 3, 1976.
- Belford, P., et. al.: "Specifications: The Key to Effective Software Development", Proc. 2nd. Int. Conf. on Soft. Eng., 1976.
- Bell, T., and T. Thayer: "Software Requirements: Are They Really a Problem?", Proc. 2nd Int. Conf. on Soft. Eng., 1976.
- Boehm, B.: "Software and Its Impact: A Quantitative Assessment", Dataamation, vol. 19, no. 5, May 1973.
- Boehm, B.: "Some Steps Towards Formal and Automated Aids to Software Analysis and Design", Information Processing 74, North-Holland, 1974.
- Brooks, F.: The Mythical Man-Month, Addison-Wesley, 1975.
- Burns, I., et. al.: "Current Software Requirements Engineering Methodology", TRW Systems Group, Huntsville, Alabama, 1974.
- Buxton, J., and B. Randall: "Software Engineering Techniques", Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 1969.

- Carter, D. E., et. al.: A Study of Critical Factors in MIS for the U. S. Air Force, NTIS no. AD-ACC9-647/9WA, Colorado State University, 1975.
- Cave, J., and A. Salisbury: "Controlling the Software Development Cycle - The Project Management Task," IEEE Trans. on Soft. Eng., vol. 4, no. 4, July 1978.
- Champine, G. A.: A Total Life Cycle Cost Model for a Computer System, PhD Thesis, University of Minnesota, 1975.
- Chen, P.: "The Entity-Relationship Model - Towards a Unified View of Data," ACM Trans. on Database Systems, vol. 1, no. 1, March 1976.
- Cooper, J.: "Corporate Level Software Management," IEEE Trans. on Soft. Eng., vol. 4, no. 4, July 1978.
- Cornell, D., and M. Woodward: "A Comparison and Evaluation of Graph Theoretical Clustering Techniques," Infor, vol. 16, no. 1, February 1978.
- Cougar, J. D.: "Evolution of Business System Analysis Techniques", Computing Surveys, vol. 5, no. 3, Sept. 1973.
- Cyert, R., and J. March: A Behavioral Theory of the Firm, Prentice-Hall, 1963.
- Davis, C., and C. Vick: "The Software Development System", IEEE Trans. Soft. Eng., vol. 3, no. 1, Jan. 1977.
- DeMarco, T.: Structured Analysis and System Specification, Yourdon Inc., 1978.
- Eds, N.: Graph Theory with Application to Engineering and Computer Science, Prentice-Hall, 1974.
- DeWolf, B.: "Requirements Specification and Design for Real-time Systems: A Problem Statement", IR&D Memo No. 4, C. S. Draper Labs., Jan. 1977.
- Dijkstra, E.: "GOTO Considered Harmful," Letter to the editor, Comm. of the ACM, vol. 11, no. 3, March 1968.
- Dijkstra, E.: A Discipline of Programming, Prentice-Hall, 1976.
- Dolotta, L., et. al.: Data Processing in 1962 - 1967, Wiley-Interscience, 1976.

- Estabrooke, G.: "A Mathematical Model in Graph Theory for Biological Classification," Journal of Theoretical Biology, vol. 12, no. 2, 1966.
- Folius, J., S. E. Madnick, and H. Schutzman: "Virtual Information in Database Systems," Report CISR-3, Sloan School of Management, M.I.T., Cambridge, Mass., July 1974.
- Ford, L., and D. Fulkerson: Flows in Networks, Princeton University Press, 1962.
- Frank, W.: "The New Software Economics," Parts 1, 2, and 3, Computerworld, Jan. 1979.
- Ginsberg, M.: "A Detailed Look at Implementation Research," MIT Sloan School Center for Information Systems Research WP 753-74, 1974.
- Goetz, M.: "The Software Products Industry - Its Future and Promise," Computerworld, In-Depth Report, October 17, 1978.
- Gottlieb, C., and S. Kumar: "Semantic Clustering of Index Terms," Journal of the ACM, vol. 15, no. 4, October 1968.
- Gutierrez, M., and U. Schirmer: "Description and Analysis of the M.I.T. Budgeting Process," MS Thesis, Sloan School of Management, Cambridge, Mass., May 1977.
- Hamilton, M., and S. Zeldin: "Higher Order Software - A Methodology for Defining Software", IEEE Trans. on Soft. Eng., vol. 2, no. 1, March 1976.
- Haney, F.: "The Architecture of Software," Data Base, ACM SIGBDP Newsletter, 1976.
- Hartigan, J.: Clustering Algorithms, John Wiley, 1975.
- Hartman, W., et. al.: Management Information Systems Handbook, McGraw-Hill, 1968.
- Heninger, K.: "Specifying Software Requirements for Complex Systems: New Techniques and Their Applications," Conf. on Specifications for Reliable Software, IEEE Computer Society (IEEE Catalog No. 79 CH1401-9C), April 1979.
- Herzog, F., and G. Matson: "The M.I.T. Budgetary Process: A Descriptive Model," Internal Report, M.I.T. Office of Administrative Information Systems, Cambridge, Mass., 1974.

Himmelblau, D.: Decomposition in Large Scale Systems, North-Holland 1973.

Hoffer, J., and D. Severance: "The Use of Cluster Analysis in Physical Database Design," Technical Report 267, Department of Operations Research, Cornell University, August 1975.

Holden, Timothy: "A Systematic Approach to Designing Complex Systems: Application to Software Operating Systems", Internal Report P010-7805-05, M.I.T. Center for Information Systems Research, May 1978.

Holton, J. B.: "Are the New Programming Technologies Being Used?", Datamation, vol. 23, no. 7, July 1977.

Horowitz, L., et. al.: Practical Strategies for Developing Large Software Systems, Addison-Wesley, May 1975.

Honeywell Inc.: Operating System/AADC: Preliminary Functional Specifications, Honeywell Systems and Research Division, Naval Air Development Center, Contract N62269-72-C-0051, 1972.

Hubert, L.: "Some Applications of Graph Theory to Clustering," Psychometrika, vol. 39, no. 3, September 1974.

Hudock, K.: "Report on User Needs for a New M.I.T. Budgeting System," Internal Report, M.I.T. Office of Administrative Information Systems, Cambridge, Mass., 1978.

Huff, S. L., and S. E. Madnick: "An Approach to the Construction of Functional Requirement Statements for System Architectural Design", Internal Report No. P010-7806-06, NTIS no. A057802, June 1978(a).

Huff, S. L., and S. E. Madnick: "An Extended Model for a Systematic Approach to the Design of Complex Systems", Internal Report No. P010-7806-07, NTIS no. A058565, July 1978(b).

Huff, S.: "Decomposition of Weighted Graphs Using the Interchange Partitioning Algorithm," Technical Report 3, MIT Sloan School of Management, January 1979(a).

Huff, S.: "Analysis Techniques for Use With the Extended SDM Model," Technical Report 9, MIT Sloan School of Management, February 1979(b).

- Huff, S.: "A Normative Cost-Benefit Analysis of the Systematic Design Methodology," Technical Report 10, Sloan School of Management, M.I.T., February 1979(c).
- Huff, S.: "Architectural Design of a New MIT Budgeting System: An Application of the Systematic Design Methodology," Technical Report 11, MIT Sloan School of Management, March, 1979(d).
- IBM: Improved Programming Technologies: Management Overview, IBM Corp., Publication GE19-5086-1, Bethesda, MD, June 1976.
- IBM: Information Management System: Virtual Storage, IBM Corp., Publication GH20-1260-1, Armonk, NY, 1974.
- Jackson, M.: Principles of Program Design, Academic Press, 1975.
- Jones, J. C.: Design Methods, Wiley-Interscience, 1970.
- Karp, R.: "On the Computational Complexity of Combinatorial Problems," Networks, vol. 5, no. 2, 1975.
- Kernighan, B., and S. Lin: "An Efficient Heuristic Procedure for Partitioning Graphs," Bell System Technical Journal, vol. 49, no. 2, 1970.
- Kustanowitz, A.: "System Life Cycle Estimation," Proc. Third Int. Conf. on Software Engineering, IEEE Publications, 1978.
- Leintz, B. et. al.: "Characteristics of Application Software Maintenance," Comm. of the ACM, vol. 21, no. 6, June 1978.
- Levine, J. H.: "The Sphere of Influence," American Sociological Review, vol. 37, no. 2, February 1972.
- Madnick, S. E., and J. Donovan: Operating Systems, McGraw-Hill 1975.
- McCabe, E.: "Locality in Logical Database Systems: A Framework for Analysis," M.S. Thesis, Massachusetts Institute of Technology, July 1978.
- McCabe, T. S.: "A Complexity Measure," IEEE Trans. on Soft. Eng., vol. 2, no. 6, December 1976.

McCormack, W. et. al.: "Problem Decomposition and Data base Reorganization Using a Clustering Technique," Operations Research, vol. 20, no. 5, September 1972.

Miller, George: "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", Psychological Review, vol. 63, no. 2, March 1956.

Mills, H. D.: "Top-Down Programming in Large Systems," in Debugging Techniques in Large Systems, (K. Rustin, ed.), Prentice-Hall, 1971.

Myers, G.: "Characteristics of Composite Design", Datamation, vol. 19, no. 9, September 1973.

Myers, G.: Composite/Structured Design, Van Nostrand Reinhold, 1978.

Naur, P., and B. Randell: Software Engineering, NATO Scientific Affairs Division, Brussels, Belgium, 1968.

Parnas, D.: "Information Distribution Aspects of Design methodology", Information Processing 71, North-Holland 1971.

Parnas, D.: "On the Criteria to be Used in Decomposing Systems into Modules", Comm. of the ACM, vol. 15, no. 12, Dec. 1972.

Peters, L., and L. Tripp: "Comparing Software Design methodologies," Datamation, vol. 23, no. 11, November 1977.

Piistranta, A.: "Resource Analysis of Computer Programming System Development," in On The Management of Computer Programming, G. Weinbaum (ed.), Auerbach Publishers, 1970.

Putnam, F.: "A General Empirical Solution to the Macro Software Estimating and Sizing Problem," IEEE Trans. on Soft. Eng., vol. 4, no. 4, July 1973.

Pyle, J.: "Hierarchies - An Ordered Approach to Systems Design," INFOTEC State of the Art Report No. 6, 1975.

Robinson, L.: "HDM - Command and Staff Overview", T.I. CSL-49, SRI Project 4628, SRI International, Menlo Park, CA., February 1978.

- Ross, D., and K. Schomann: "Structured Analysis for Requirements Definition", IEEE Trans. Soft. Eng., vol. 3, no. 1, Jan. 1977.
- Sage, A.: Methodology for Large Scale Systems, McGraw-Hill, 1977.
- Salton, G., and A. Wong: "Generation and Search of Clustered Files," ACM Trans. on Database Systems, vol. 3, no. 4, December 1978.
- Sanjiovanni-Vincentelli, A., et. al.: "An Efficient Heuristic Cluster Algorithm for Tearing Large-scale Networks," IEEE Trans. on Circuits and Systems, vol. 24, no. 12, December 1977.
- Scott, K., and D. Simmons: "Predicting Program Development Productivity - A Communications Model," Proc. First Nat. Conf. on Soft. Eng., IEEE Publications, Sept 1975.
- Silver, A.: "A Computer Analysis Tool for Structural Decomposition Using Entropy Metrics," Internal Report, Martin-Marietta Corp., Denver, Colorado, 1978.
- Simon, H.: The Sciences of the Artificial, MIT Press, 1969.
- Spillers, W. R.: Basic Questions in Design Theory, North-Holland, 1974.
- Spillers, W. R.: "Design Theory", IEEE Trans. on Systems, Man, & Cybernetics, March 1977.
- Stay, J. F.: "HIPO & Integrated Program Design", IBM System Journal, vol. 5, no. 2, June 1976.
- Stevens, J., et. al.: "Structured Design", IBM Systems Journal, vol. 13, no. 2, April 1974.
- Strang, G.: Linear Algebra and Its Applications, Academic Press, 1976.
- Taqqart, W., and M. Tharp: "Survey of Information Requirements Analysis Techniques", Computing Surveys, vol. 9, no. 4, 1977.
- Teichroew, D., and H. Sayari: "Automation of System Building", Datamation, vol. 17, no. 8, 1971.

Teichroew, I.: "A Survey of Languages for Stating Requirements for Computer-based Information Systems", Proc. of the Fall Joint Computer Conference, AFIPS Press, 1972.

Teichroew, I., and M. Bastarache: "PSL Users' Manual", SDOS TR No. 98, March 1975.

Teichroew, I., and E. Hershey: "PSL/PSA: A Computer-aided Technique for Structured Documentation and Analysis of Computer-based Information Systems", IEEE Trans. Soft. Eng., vol. 3, no. 1, Jan. 1977.

Thayer, T.: "Understanding Software Through Empirical Reliability Analysis," Proc. of the NCC, 1975.

Uhrig, J.: "Systems Requirements Specification for Real-time Systems," Bell Telephone Labs., Whippany, NJ, 1978.

Vick, C., and C. Davis: "Requirements: Software Engineering's Big Hurdle," Ballistic Missile Defence Advanced Technology Center, Huntsville, Alabama, 1978.

von Letkemann, H., B. Shaw, et. al.: "Budget System Functional Requirements," and "Supplement to Functional Requirements," Internal Reports, M.I.T. Office of Administrative Information Systems, September 1978.

Ward, J.: "Hierarchical Grouping to Optimize an Objective function," American Statistical Association Journal, March 1963.

Wasserman, I., et. al.: "Software Engineering: The Turning Point", Computer, September 1978.

Wasserman, I.: "Toward the Engineering of Software: Problems of the 1980's," The Oregon Report, IEEE Computer Society, 1978.

Watson, G., and L. Felix: "Method of Program Measurement and Estimation," IBM Systems Journal, vol. 16, no. 1, 1977.

Weinberg, G.: The Psychology of Computer Programming, Van Nostrand Reinhold, 1971.

White, J., and T. Booth: "Towards an Engineering Approach to Software Design", Proc. 2nd Int. Conf. on Soft. Eng., 1976.

Wilson, M.: "The Information Automat Approach to Design and Implementation of Computer-based Systems," Report FSD 76-0093, IBM Corp., Armonk, New York, 1976.

Wirth, N.: "program Development by Stepwise Refinement", Comm. of the ACM, April 1971.

Wirth, N.: Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

Wolverton, R.: "The Cost of Developing Large-Scale Software," IEEE Trans. on Computers, vol. 23, no. 6, June 1974.

Yau, S., et. al.: "Ripple Effect Analysis of Software Maintenance," Trans. Third Int. Conf. on Soft. Eng., IEEE Publications, 1978.

Appendix A

COMPLETE LISTING OF PSL STATEMENT TYPES.

A.1 System Flow Statements.

- (1) process/interface RECEIVES input/output
 - (1-a) input/output RECEIVED BY process/interface.
- (2) interface RESPONSIBLE FOR set.
 - (2-a) set RESPONSIBLE-INTERFACE interface.
- (3) process/interface GENERATES input/output
 - (3-a) input/output GENERATED BY process/interface.

A.2 System Structure statements.

- (4) input/output/process/interface PART OF input/output/process/interface.
 - (4-a) input/output/process/interface SUBPARTS ARE input/output/process/interface.
- (5) entity/input/output CONTAINED IN set/set/set
- (6) set SUBSET OF set.
 - (6-a) set SUBSETS ARE set.
- (7) set/set/set SUBSETTING-CRITERIA ARE subsetting-criteria/element/group.

(7-a) subsetting-criteria/element/group
SUBSETTING-CRITERION set/set/set.

(8) process UTILIZED BY process.

(8-a) process UTILIZES process.

(9) interval CONSISTS OF interval (optionally preceded by
system parameter).

A.3 Data Structure.

(10) input/output/entity/set/group CONSISTS OF group or
element/" "/" "/"

(11) entity IDENTIFIED BY group/element.

(11-a) group/element IDENTIFIES entity.

(12) entity RELATED TO entity VIA relation.

(12-a) relation BETWEEN entity AND entity.

(13) relation ASSOCIATED-DATA IS element/group.

(13-a) group/element ASSOCIATED WITH relation.

A.4 Data Derivation.

(14) input/entity/set/group/element USED BY pro-
cess/" "/" "/"

(14-a) process/" "/" "/" USES
input/entity/set/group/element.

(15) input/entity/set/group/element USED BY process TO
DERIVE entity/set/element/group/output.

(15-a) process USES input/entity/set/group/element TO DERIVE output/entity/set/group/element.

(16) input/entity/set/group/element USED BY process TO UPDATE entity/set/group/element.

(16-a) process USES input/entity/set/group/element TO UPDATE entity/set/group/element

(17) output/entity/set/group/element DERIVED BY process.

(17-a) process DERIVES output/entity/set/group/element.

(18) output/entity/set/group/element DERIVED BY process USING input/entity/set/group/element.

(18-a) process DERIVES output/entity/set/group/element USING input/entity/set/group/element.

(19) entity/set/group/element UPDATED BY process.

(19-a) process UPDATES entity/set/group/element.

(20) entity/set/group/element UPDATED BY process USING input/entity/set/group/element.

(20-a) process UPDATES entity/set/group/element USING input/entity/set/group/element.

(21) set/relation DERIVATION comment-entry.

(22) relation/defined-name MAINTAINED BY process.

(22-a) process MAINTAINS relation, subsetting-criterion.

(23) process PROCEDURE comment-entry.

A.5 System Size.

(24) entity/set/relation CARDINALITY IS system-parameter.

(25) relation CONNECTIVITY IS system-parameter TO system-parameter.

(26) element/defined-name VALUE IS positive-integer.

(27) element/devined-name VALUES ARE minimum-value THRU maximum-value.

(28) process HAPPENS system-parameter TIMES-PER interval.

A.6 System Dynamics.

(29) input/output/event HAPPENS system-parameter TIMES-PER interval.

(30) entity VOLATILITY comment-entry.

(31) set VOLATILITY-SET comment-entry.

(32) set VOLATILITY-MEMBER comment-entry.

(33) process TRIGGERED BY event.

(33-a) event TRIGGERS process.

(34) process INCEPTION-CAUSES event.

(34-a) event ON-INCEPTION process.

(35) process TERMINATION-CAUSES event.

(35-a) event ON-TERMINATION process.

(36) condition BECOMING TRUE/FALSE CALLED event.

(36-a) event WHEN condition BECOMES TRUE/FALSE.

(37) condition TRUE/FALSE WHILE comment-entry.

A.7 Project Management.

(38) interface/input/output/entity/set/relation/ process/group/element/interval/condition/event/memo/defined-name RESPONSIBLE-PROBLEM-DEFINER person-name.

(38-a) person-name RESPONSIBLE FOR list as in (38).

(39) problem-definer-name MAILBOX mailbox-name.

A.3 System Properties.

Note: in the following, "list" refers to the list of object types used in (38) above.

(40) list SYNONYM synonym-name(s).

(41) list DESCRIPTION comment-entry.

(42) list (excluding "memo") SEE-MEMO memo-name.

(43) list KEYWORD keyword-names.

(44) list ATTRIBUTES attribute-name(s).

(45) list SECURITY security-name(s).

(46) list SOURCE source-name(s).

(47) list (excluding, "memo", "defined-names") APPLIES name(s).

Appendix B

THE FULL SET OF DBMS REQUIREMENTS USED BY ANDREU.

1. Data base schema (data dictionary) including inter-file relationships, is defined and maintained independently of database usage.
2. Separate files can be defined to be interrelated.
3. Data description language is English-like and self-documenting.
4. Database schema is validated by system prior to usage.
5. Interfile relationships can be defined at run time.
6. Field definition permits validation of input datum as to acceptable values.
7. The maximum number of files in the data base is at least 10.
8. Maximum number of interrelated files is at least 5.
9. Maximum size of a logical record is at least 500 information characters.
10. Maximum size of an item (field) is at least 100 characters.
11. Maximum number of items in a record is at least 50.
12. Repeated fields (multi-values attributes) can be defined.
13. Variable-sized fields can be defined.
14. Records in the database can be added.
15. Records in the database can be changed.

16. Records in the database can be deleted.
17. Database update can be performed by on-line user through query language.
18. Database maintenance can be performed by batch.
19. A bulk database update (initialization) can be performed through system utility.
20. Null-value field identification and generation supported.
21. Field update can trigger computation of a correlated field in same record.
22. Field update can trigger computation (e.g., tally of a correlated field in a different record).
23. Data integrity supported at least at file level lockout on update.
24. Record level lockout.
25. Checkpoint/restore facilities.
26. Transaction history facility.
27. Separate security facilities for retrieval and update.
28. Database level security.
29. File level security.
30. Field-level security.
31. Non-procedural user's language available for on-line query and update.
32. Boolean conditionals can be used in record-selection criteria.
33. Relational conditionals can be used in record selection criteria.
34. Arithmetic expressions can be used in record selection criteria.
35. Text-string scanning expressions can be used in record selection criteria.

36. Relational condition can compare variable to variable.
37. Data meeting selection criteria can be used for subsequent query processing.
38. Selected data can be stored by at least one sort key.
39. Capability for report formatting.
40. Report formatting optionally automatic.
41. Report break control feature supported.
42. Report summary line feature supported.
43. Multi-valued fields can be selectively listed.
44. Screen (menu) formatting facilities supported.
45. Local keyboard terminal supported.
46. Remote keyboard terminal supported.
47. CRT supported.
48. Delat Data 5260 supported.
49. User can interrogate status of system.
50. User can interrogate status of current report.
51. User can cancel active request without loss of data integrity.
52. User can suppress listing, save report, and later re-initiate listing.
53. User can direct output to system printer.
54. User can route listing to other terminal.
55. Capability to broadcast messages to all terminals.
56. Signon security.
57. A master terminal facility with privileged commands and control is supported.
58. Master terminal can be relocated to any on-line terminal.

59. System set-up effort, and each subsequent sysgen, less than one man-month.
60. Utilities to aid set-up.
61. Average up-time for minimum configuration at least 95 percent over a 30-day period.
62. Average system recovery time is 2 hours over a 30-day period.
63. Maximum recovery time is 24 hours.
64. Maintenance requirements less than 1 hour per week.
65. Power fail restart facility.
66. Dual processor fail-soft capability.
67. Removable disks containing data base can be mounted and processed during an on-line session.
68. A job accounting recording facility is supported sufficient to charge users by application and by department.
69. A job accounting reporting facility.
70. Application is transportable to/from Agency's existing systems.
71. Data is transportable to/from Agency's existing systems.
72. System can operate in ordinary office environment.
73. System can communicate with other Agency's systems.
74. With a single terminal active, user can receive a response from a direct access to any item in the data base in less than 5 seconds.
75. Response time as independent as possible of file size.
76. Capability to support at least 10 active terminals.
77. Capability to support two or more concurrent queries in different stages of processing.

78. Display rate of terminal at least 120 characters per second on a CRT and 15 characters per second on a hard-copy terminal.
79. Dynamic file reorganization capability.
80. Dynamic reallocation of released and deleted storage areas.
81. File size limited only by disk storage capacities.
82. Total on-line data base can be distributed over many disk units.
83. Controls for tuning system performance at sysgen or system load time.
84. Controls for dynamically tuning system performance during run time.
85. Application tuning can be accomplished by restructuring the data to bias retrieval vs. update performance characteristics.
86. Can be delivered within M1 months.
87. Being used by at least 10 users within M2 months.
88. Non-recurring costs for basic configuration not more than C.
89. Maintenance costs less than MC per month.

Appendix C

EDITED DBMS REQUIREMENTS TRANSFORMED TO TEMPLATE FORM.

The number in brackets refers to the requirement number used by Andreu (see Appendix B).

C.1 Existence Statements.

General template form:

There will be <mod> <object>.

1. (2) There will be file interrelationships.
2. (12) " Repeated field definitions.
3. (25) " checkpoint/restore facilities.
4. (26) " transaction history facilities.
5. (28) " database level security facilities.
6. (29) " file level security facilities.
7. (30) " field level security facilities.
8. (31) " non-procedural query/update language.
9. (39) " report formatting facility.
10. (41) " report break control facility.
11. (42) " report summary line facility.
12. (44) " menu formatting facility.
13. (45) " local keyboard terminal operation.
14. (46) " remote keyboard terminal operation.
15. (47) " CRT keyboard terminal operation.

- 16. (48) " Delta Data 526C terminal operation.
- 17. (53-a) " system printer.
- 18. (55) " message broadcast facility.
- 19. (56) " signon security facility.
- 20. (57) " master terminal facility.
- 21. (57-b) " master terminal privileged commands.
- 22. (60) " set-up assistance utilities.
- 23. (65) " power fail restart capability.
- 24. (66) " dual-processor fail-soft capability.
- 25. (68-a) " job accounting recording facilities.
- 26. (69) " job accounting reporting facilities.
- 27. (73) " inter-system communication facilities.
- 28. (83-a) " static tuning controls.
- 29. (84-a) " dynamic tuning controls.

C.2 Property template statements.

General form for property template:

<mod> <object> can/will be <mod> <property>.

- 30. (3-a) Data definition language will be English-like.
- 31. (3-b) Data definition language will be self-documenting.
- 32. (20-a) Data fields can be null.
- 33. (20-b) Null fields will be identifiable.
- 34. (40) Report formatting facilities will be optionally automatic.
- 35. (49) System status will be queryable.

- 36. (50) Current request status will be queryable.
- 37. (67-a) Database disk units will be dismountable.
- 38. (70) Application programs will be transportable between existing systems.
- 39. (71) Data files will be transportable among existing systems.
- 40. (82) On-line databases can be distributed across multiple disks.
- 41. (85) Logical files can be alternately physically organizable.
- 42. (13) Data fields can be variable-sized.

C.3 Treatment template statements.

General form for treatment template is:

<mod> <object> can/will be <mod> <treatment>.

- 43. (6) Data items can be validated using field definition information.
- 44. (14) Database can be increased.
- 45. (15) Database records can be changed.
- 46. (16) Database records can be deleted.
- 47. (17-b) Database records can be updated using query language.
- 48. (18) Database records can be maintained using batch processing.

49. (19-a) Database can be initialized using system utility.
50. (19-b) Database can be updated using system utility.
51. (23) Files can be locked during update.
52. (24) Records can be locked.
53. (38-a) Selected data can be sorted.
54. (43) Multi-values fields can be listed selectively.
55. (51-a) Active requests can be cancelled.
56. (51-b) Data integrity will be maintained with respect to active request cancellation.
57. (52-a) Report listing can be suppressed.
58. (52-b) Suppressed report listing can be saved.
59. (52-c) Saved report listing can be re-initialized.
60. (53-b) Output can be printed using system printer.
61. (54) User-produced listing can be printed using alternative terminals.
62. (58) Master terminal can be relocated using alternative on-line terminals.
63. (79) Files can be re-organized dynamically.
64. (80) Released memory can be re-allocated dynamically.

C.4 Volume Template Statements.

General form for Volume Templates

<mod> <object> can/will be <order statement> <measure>.

65. (59-a) System set-up effort will be less than 1 man-month.
66. (59-b) System sysgen effort will be less than 1 man-month.
67. (61) Minimum configuration 30-day average up-time will be at least 95 %.
68. (62) Average 30-day recovery time will be no more than 2 hours.
69. (63) Maximum recovery time will be no more than 24 hours.
70. (64) Weekly maintenance time will be less than 1 hour.
71. (74) Single-active-terminal direct-access response time will be less than 5 seconds.
72. (76) Active terminal number can be at least 10 units.
73. (77) Concurrent query number can be at least 2 units.
74. (78-a) CRT display rate can be at least 120 characters per second.
75. (78-b) Hardcopy display rate can be at least 15 characters per second.
76. (81) File size can be equal to available physical storage size.
77. (86) Development time will be no more than M1 months.
78. (87) Ten-user installation time will be no more than M2 months.

79. (88) Basic configuration non-recurring costs will be no more than C dollars.
80. (89) Maintenance costs will be no more than MC dollars/month.
81. (7-b) Maximum number of database files can be at least 10 units.
82. (8-b) Maximum number of interrelated database files can be at least 5 units.
83. (10-b) Maximum number of field characters can be at least 100 units.
84. (9-b) Maximum number of logical record characters can be at least 500 units.
85. (11-b) Maximum number of logical record fields can be at least 50 units.
86. (38-b) Number of possible sort keys will be at least 1 unit.

C.5 Timing template statements.

General form for timing statement template is:

<mod> <object> can/will be <timing relationship>

<mod> <object>.

87. (4) Schema validation will occur before database usage.
88. (5) Inter-file relationship usage can occur before database usage.

- 89. (17-a) Database update can occur before database on-line usage.
- 90. (21) Field update can trigger same-field-other-record update.
- 91. (22) Field update can trigger computation.
- 92. (67-b) Disk mounting can occur during on-line database usage.
- 93. (83-b) Static tuning can occur after database loading.
- 94. (83-c) Static tuning can occur after sysgen.
- 95. (84-b) Dynamic tuning can occur during database usage.

C.6 Relationship template statements.

General form for Relationship Subsetting template:

<mod> <object> can contain <mod> <object>.

- 96. (1-a) Schema definition can contain inter-file relationships.
- 97. (7-a) Database can contain files.
- 98. (8-a) Database can contain inter-related files.
- 99. (9-a) Logical records can contain characters.
- 100. (10-a) Fields can contain characters.
- 101. (11-a) Logical records can contain fields.
- 102. (32) Record selection criteria can contain boolean conditional expressions.
- 103. (33) Record selection criteria can contain relational conditional expressions.

104. (34) Record selection criteria can contain arithmetic expressions.
105. (35) Record selection criteria can contain text-string scanning expressions.
106. (36) Relational conditional expressions can contain variable-to-variable comparisons.
107. (68-b) Job accounting data will contain application charges.
108. (68-c) Job accounting data will contain organizational unit charges.

General form for Relationship Independence template:

<mod> <object> can/will be independent of <mod>
<object>.

109. (75) Response time will be (as much as possible) independent of file size.
110. (1-b) Schema definition will be independent of database usage.
111. (27) Retrieval security privileges will be independent of update security privileges.

Appendix D

SDM ANALYSIS PACKAGE DOCUMENTATION.

The analysis program developed for decomposing and analyzing SDM graphs is written in IBM PL/1, and operates within the VM/370-CMS environment. The package is presently run on the MIT 370/168 computer system. While the package is still being tested, and may be slightly modified in the future, it is stable enough in its general characteristics to warrant some brief documentation, as provided here.

General features of the analysis package include:

1. it has been implemented according to the general precepts of structured programming; all subprograms are constrained in size, and have relatively simple structure;
2. the driving routines are command-driven, hence easily extendable (e.g., additional new types of decomposition algorithms could be easily added to the system);
3. subprograms are functionally specific; the relationship between caller and calling routines is always clear and easy to understand;
4. effective use of PL/1's powerful data types has been made in representing clearly and succinctly the key databases used within the system.

The package consists of a master program (named, naturally, MASTER), together with a set of subprograms to implement the various functions. The MASTER program is command-driven, so the user executes various commands by typing the command name (followed in some cases by additional information, after secondary prompting by the system). The various commands, which will be explained in more detail shortly, are:

<u>COMMAND</u>	<u>FUNCTION</u>
STOP	Terminate session, return to the CMS environment;
READGRAP	Read structure information for a graph to be analyzed, from a previously established disk file;
PRINTADJ	Print the adjacency matrix for the graph to the terminal or to the line printer;
CALCSIM	Calculate a similarity matrix for the current graph;
PRINTSIM	Print the similarity matrix to the terminal or the line printer;
SAVESIM	Write the similarity matrix to a disk file, for later retrieval;
READSIM	Read a previously saved similarity matrix from a disk file;
CLUSTER	Execute one of the four hierarchical clustering routines on the current similarity matrix;
READCLUS	Read the trace of the clustering from the file written during the execution of the CLUSTER command;
PRINTMEA	Calculate and type at the terminal or the line printer the goodness

measure for specified stages in the clustering trace;

PRINTCLU Type at the terminal or line printer the node clusters for certain specified stages in the clustering trace;

MODIFY Make incremental changes to the clustering at a certain stage in the clustering trace;

INTERCH Execute the interchange partitioning algorithm (a series of subcommands are issued).

A control feature built into the analysis package checks the logical consistency of each command. For example, attempting to execute a **SAVESIM** or **PRINTSIM** prior to having calculated a similarity matrix (i.e., to having issued a **CALCSIM** command) causes a status error. A status error message is issued, and execution continues.

D.1 MORE ON THE MASTER COMMANDS.

We consider now each of the above commands, in somewhat more detail.

(1) **STOP.** This command requires no additional explanation. However, it should be noted that the master program has an attention interrupt trap: hitting the "attention" (or

"break") key causes the program to stop its current activity and request a new command. Thus STOP is really the only way of "gracefully" exiting from the MASTER routine (the other way being to hit the attention key multiple times, causing a forced transfer to the CP environment, not a recommended practice).

(2) READGRAP. A typical analysis session begins with reading in a particular graph structure via the READGRAP command. The graph data must have been previously set up in a standard CMS disk file. The first entry must be the number of nodes in the target graph. The remaining entries are to be of the form

(n1,n2,weight),

to indicate that the nodes n1 and n2 are linked in the requirements graph by an interdependency with an associated strength value of "weight." Thus an illustrative graph structure file for a 6-node graph might appear as:

```
6
1 2 .5
1 3 .3
1 4 .8
3 4 .5
2 5 .3
2 6 .4
```

5 6 .8

The order of n1 and n2 in a particular entry is immaterial. The entries need not be on separate logical lines in the file, although in practice it is easier to enter them that way. The graph structure file, as for all files used by the package, must be given a CMS name; for instance, the name "GRAPH DATA A" might be used to identify this file to CMS, whereas the name "GRAPH" could be used for referencing the same file within the PL/1 routines.

Other files that play a role in the execution of MASTER include: (a) the file on which the similarity matrix may be saved for later use (this file is currently named "TEMP-SIM"); (b) the intermediate file on which the clustering trace is written during execution of the CLUSTER command (named CLTRACE); (c) a file named "OPTCLUS" on which an optimum cluster vector may be written for later use by other standalone routines.

(3) PRINTADJ and PRINTSIM. These commands use a common sub-program to print out either the adjacency or similarity matrices, either to the user's terminal or to the high-speed line printer located in the main computer facility. The particular device to be used (terminal, or line printer) is determined by a code digit that the user types in response to a follow-up system prompt message.

The matrices are labelled, and the rows and columns numbered appropriately. If the matrix is larger than 15x15, it is "folded" column-wise - i.e., the first 15 columns are printed, followed by the next 15, etc. In all cases, since both matrices are symmetrical, only the lower triangular form is printed.

(4) CALCSIM. This command causes the system to execute a subroutine that calculates all the elements of the similarity matrix, given the weighted adjacency matrix. The calculation is based on the algorithm described in Section 2.2.

(5) SAVESIM, READSIM. These commands write and read, respectively, a temporary copy of the similarity matrix to or from a disk file set up for this purpose. This file is named TEMPSIM within the MASTER routine. These commands make it possible to save the results of a similarity calculation from one session, to be used in a later session that involves the same graph, thereby avoiding the cost of recalculating the entire matrix.

(6) CLUSTER. This command executes one of the four hierarchical clustering routines described earlier. A secondary prompt requests the user to enter the appropriate "version" number - i.e., "1" for "HIER1", etc. Recall that:

HIER1 - Single linkage,
HIER2 - Complete linkage,
HIER3 - Maximum pre-merge centroid,
HIER4 - Maximum post-merge centroid.

The clustering routine writes a "trace" - that is, a record of the nodal clustering at each step - to an intermediate file named "CLTRACE." As with SAVESIM, this is done in order to provide a "restart" capability for extended or interrupted analysis sessions.

(7) READCLUS. This command reads the clustering trace from the intermediate file "CLTRACE." This can be used to initialize a previously generated clustering trace for further analysis. Of course, it cannot be used until at least one CLUSTER command has been executed, either in the current or an earlier session. It is generally necessary to execute a READCLUS command following each execution of the CLUSTER command; failing to do so may result in performing analysis upon the clustering trace from an earlier CLUSTER calculation.

(8) PRINTMEA. This command can be used to calculate the goodness measure M for one or a consecutive series of steps in the currently active clustering trace. The actual pass

or passes, as well as the output device to be used (terminal or line printer) are entered following secondary promptings.

For example, suppose a 40-node graph had been CLUSTERed, giving rise to a clustering trace consisting of 40 steps, or "passes." The first pass corresponds to each node as a separate cluster; the 40th pass corresponds to a single cluster containing all 40 nodes. In response to a secondary prompt, the user enters the "frompass" and "topass" values (the system checks logical consistency). If only a single pass's measure is required, the user enters that pass number twice in succession. The system responds by calculating and printing out the goodness measure M for each specified pass in the clustering trace, in the form

PASS = nn MEASURE = mm.mmm .

(9) PRINTCLU. This command operates in a manner similar to PRINTMEA. However, the clusters themselves, rather than the goodness measures, are printed. A typical printout might appear as:

```
* * * PASS = 8 * * *  
CLUSTER 1:  5  6  7  
CLUSTER 2:  1  2  3  4  
CLUSTER 3:  8  9 10
```

As with PRINTMEA, the user is prompted for device type and for range of passes in the trace for which the clustering information is to be printed.

(10) MODIFY. This command allows the user to make incremental changes to a current clustering in order to search for local improvements or test out ideas for better decompositions. Following a secondary prompt, the user specifies which pass in the clustering trace he wishes to modify, and the nature of the modifications (which nodes to be placed in which clusters). The resulting decomposition may then be measured, printed, or saved on a special ("OPTCLUS") disk file for later use.

(11) INTERCH. This last command transfers control to the interchange partitioning routine, described next.

D.2 INCHCTL (INTERCHANGE CCONTROL) COMMANDS.

Most of the primary commands in the analysis package are concerned with setting up the graph data, printing out various data, and executing the clustering routines. This command, however, serves only to pass control to a major subprogram (INCHCTL). This subprogram plays a role somewhat similar to MASTER itself, but with respect to the interchange algorithm.

The interchange technique is described in detail in Chapter 6, and its operational aspects will not be discussed here.

The routine INCHCTL accepts a set of subcommands from the terminal, which allow the user to control step by step the execution of the interchange algorithm. This control is especially useful in performing certain sensitivity analyses during the course of the algorithm's execution.

The commands available within INCHCTL include:

<u>COMMAND</u>	<u>FUNCTION</u>
RETURN	Return control to MASTER;
INITIAL	Initialize the current and proposed partitions;
TYPECUR	Print the current partition at the user's terminal;
TYPEPRC	Print the proposed partition at the user's terminal;
EVALCUR EVALPRC	Evaluate the current/proposed partitions and print the M value at the terminal;
UPDATECU	Replace the current partition with the proposed partition;
RESETPRO	Reset the proposed partition to the value of the current partition;
CALCSTR	Calculate the strength of a specific, or of all, subgraphs in the current partition;
SPLIT	Call the interchange algorithm to partition a particular subgraph in the current decomposition, returning the result in the proposed partition.

AUTO

Execute a master control algorithm to automatically decompose the entire graph.

A central concept in the INCHCTL routine is the use of a "current" and a "proposed" partition. The reason for having two potentially different active partitions is to allow tentative actions to be taken in the partitioning process without making them irrevocable. For instance, a user might wish to try partitioning a particular subgraph under various minimum-size sub-partition constraints, then select the particular version giving the best M value. Since it would be an extreme computational burden to try all possible combinations, INCHCTL is set up to take advantage of the user's ability to "feel" for a good alternative with a few trial-and-error attempts.

While most of the above commands are self-explanatory, a couple deserve further elaboration.

(1) INITIAL. The structure of the target graph is passed to INCHCTL via the parameter list in the subroutine call. However, no information regarding the initialization of the current and proposed partitions is available at the start. Thus the user's first task is to establish an appropriate decomposition initialization. Normally, one would want to begin at the beginning - i.e., with the entire graph treated

as a single "subgraph." Alternatively, a user may have some particular partitioning in mind that he would like to "try out," or from which he would like to start the analysis. Thus the INITIAL command produces a secondary prompting message asking the user to choose between initializing the current and proposed partitions "from the beginning," and entering some other initialization of his choice. Format requirements under the latter option is provided in the prompting message.

(2) CALCSTR. This command may be used to decide which subgraph within the current decomposition to attack next. Since it is often the case that a user will wish to calculate the strength of all the current subgraphs, a secondary prompt asks him to select a specific subgraph or to specify "all the subgraphs."

Normally, as discussed in Chapter 6, Section 6.6, it would make most sense to choose the subgraph with the lowest strength to partition next, although the user may wish to try other alternatives (e.g., the largest subgraph) in some cases.

(3) SPLIT. This command produces two secondary prompts. The first asks the user to enter the identification index of the subgraph he wishes to partition (as printed out by TYPE-

CUR); the second asks for the minimum desired subgraph size for the two subgraphs that will be produced by the interchange algorithm.

(4) AUTO. This command invokes the automatic master control procedure for stepping the interchange algorithm through an entire decomposition. A single secondary prompt asks the user to enter the minimum subgraph size (nmin). The program then proceeds to decompose the target graph by always selecting for the next split that subgraph with minimum strength and cardinality not more than twice the value of nmin.

Appendix B includes an illustration of the use of the interchange control procedure and the interchange algorithm.

D.3 ROUTINES INCLUDED IN THE ANALYSIS PACKAGE.

There are presently a total of 26 individual programs that make up the analysis package. Furthermore, a total of six different files (counting terminal and line printer as "files") may be read or written (or both) during a session. The logical relationships among the programs and files is illustrated in Figure D.1.

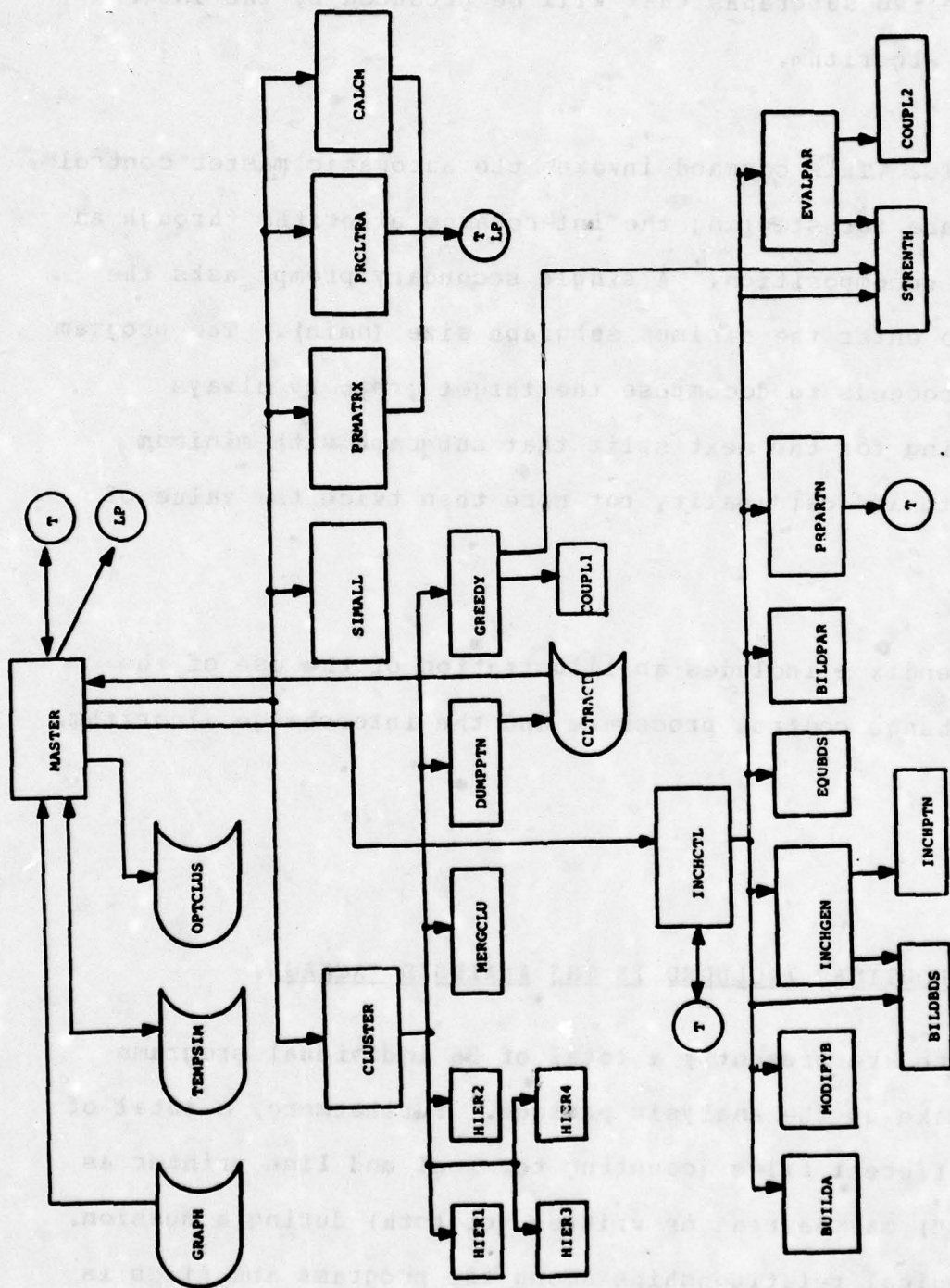


Figure D.1
Control structure for the modules of the
SDM analysis package

The purpose of each program and file is briefly stated below.

D.3.1 Programs.

- (1) MASTER. Command-driven master control routine (discussed earlier).
- (2) SIMALL. Calculates the similarity matrix.
- (3) PRMATRX. Prints the similarity or adjacency matrices to line printer or terminal.
- (4) CLUSTER. Controls the execution of the clustering algorithm.
- (5) DUMPPTN. Called by CLUSTER to write the clustering trace to the intermediate file "CLTRACE."
- (6) CALCH. Calculates and prints the decomposition goodness measure for a given step in the clustering trace.
- (7) EVALPAR. Calculates the decomposition goodness measure.
- (8) STRENGTH. Calculates the internal strength of a subgraph.
- (9) COUPL2. Calculates the coupling index between two specified subgraphs.
- (10) PRCLTRA. Prints the node clustering for a given step in the clustering trace.
- (11) H1EF1. Performs hierarchical clustering using single linkage.
- (12) H1ER2. Performs hierarchical clustering using complete

- linkage.
- (13) HIER3. Performs hierarchical clustering using largest pre-merge centroid.
 - (14) HIER4. Performs hierarchical clustering using largest post-merge centroid.
 - (15) GREEDY. Performs hierarchical clustering following the "greedy" algorithm (see Section 3.3).
 - (16) COUPL1. Calculates the coupling index between a particular subgraph and all other subgraphs that connect to it (for use in the "GREEDY" calculation).
 - (17) MERGCLU. Merges two specified subgraphs together.
 - (18) INCHCTL. Controls execution of the interchange algorithm (see Appendix A.2).
 - (19) BILDBDS. Converts from the vector form for storing partition information to the structure form.
 - (20) EQUEDBS. Equates one partition database to another (used by the UPDATTCU and RESETPRO commands within INCHCTL).
 - (21) BILDPAL. Converts from structure form for storing partition information to the vector form.
 - (22) BUILDPA. Creates a temporary adjacency matrix from a specified subgraph.
 - (23) MODIFYB. Updates the structure form to reflect a particular cluster merge decision.
 - (24) PRPAFTN. Prints clustering data to terminal or line printer.
 - (25) INCHGEN. Generates different starting partitions for use in

the interchange algorithm routine.

- (26) INCHPTN. Performs the interchange calculations to partition a given subgraph.

D.3.2 Files.

- (1) User's terminal.
- (2) Line printer. Data sent to the line printer is sent directly, as opposed to having it written to a disk file for later spooling. This feature can be easily modified by changing a line in the driving EXEC.
- (3) GRAPH. This is the name used within the programs for the file containing the graph structure data. Format for this data was explained in Appendix A.1.
- (4) TEMPSIM. This file is used to store the temporary similarity matrix if desired.
- (5) CLTRACE. The clustering trace is written to this file, and must be read in before executing commands dealing with the results of a particular clustering.
- (6) OPTCLUS. During the execution of the MODIFY command, the user is asked whether he wishes to save the modified decomposition he has created. If he elects to save it, it will be written to this file.

Appendix E
TERMINAL EXECUTION TRACE.

This appendix consists of the terminal execution trace of a sample session using the SDM analysis package. The graph being analyzed is the 22-node DBMS referred to in Section 4 (see Figure 4.1).

The commentary in italics was added to amplify and explain the trace data.

Explanatory Comments

MASTER *Execute the MASTER program*

EXECUTION BEGINS...

OK: .READGRAP *Read graph data from disk*

GRAPH DATA ENTERED. NNODE= 22

OK: .PRINTADJ *Print the adjacency matrix to the line printer*

PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.0

OK: .WRONGCOM *Unrecognised command*

BAD COMMAND.

OK: .CALCSIM *Calculate the similarity matrix and print it at the terminal*

OK: .PRINTSIM

PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.1

SIMILARITY MATRIX:

	1	2	3	4	5	6	7
1	1.000						
2	0.281	1.000					
3	0.315	0.450	1.000				
4	0.044	0.000	0.000	1.000			
5	0.256	0.327	0.360	0.000	1.000		
6	0.295	0.151	0.134	0.031	0.092	1.000	
7	0.093	0.141	0.188	0.000	0.256	0.000	1.000
8	0.000	0.000	0.000	0.000	!		

ATTENTION INTERRUPT.

Printout halted using
break key

OK: .CLUSTER

SELECT PROCEDURE - 1/2/3...

Perform hierarchical
clustering using HIER1

:

.1

OK: .PRINTMEA

STATUS ERROR. CLUSTERING TRACE NOT YET ESTABLISHED.

OK: .READCLUS

Clustering trace must
be read from disk prior
to analyzing

OK: .PRINTMEA

SELECT RANGE, LOWER TO HIGHER.

:

.18 22

PRINT TO TERMINAL(1) OR LINE PTR(0)?

:

.1

PASS= 18 MEASURE= -0.1487

Measures for various
stages in the clustering.
Best M = 0.0373, corres-
ponding to a single
cluster.

PASS= 19 MEASURE= -0.0996

PASS= 20 MEASURE= -0.1870

PASS= 21 MEASURE= 0.0280

PASS= 22 MEASURE= 0.0373

OK: .CLUSTER

Repeat using HIER2

SELECT PROCEDURE - 1/2/3...

:

.2

OK: .READCLUS

OK: .PRINTMEA

SELECT RANGE, LOWER TO HIGHER.

:

.16 22

PRINT TO TERMINAL(1) OR LINE PTR(0)?

:

.1

PASS= 16 MEASURE= -0.7122

PASS= 17 MEASURE= -0.6049

PASS= 18 MEASURE= 0.0602

PASS= 19 MEASURE= 0.2802

*Best measure at pass 19
(four clusters).*

PASS= 20 MEASURE= 0.1234

PASS= 21 MEASURE= 0.0280

PASS= 22 MEASURE= 0.0373

OK: .PRINTCLU

SELECT RANGE, LOWER TO HIGHER.

:

.19 19

*Examine the clustering at
pass 19.*

PRINT TO TERMINAL(1) OR LINE PTR(0)?

:

.1

*** PASS 19 ***

CLUSTER	1:	1	2	3	5	6	7	15
CLUSTER	2:	4	16	17	18	21	22	
CLUSTER	3:	8	9	10	11	12	19	20
CLUSTER	4:	13	14					

OK: .CLUSTER

*Repeat clustering using
HIER3*

SELECT PROCEDURE - 1/2/3...

:

.3

OK: .READCLUS

OK: .PRINTMEA

SELECT RANGE, LOWER TO HIGHER.

:

.18 22

PRINT TO TERMINAL(1) OR LINE PTR(0)?

:

.1

PASS= 18 MEASURE= 0.2958

PASS= 19 MEASURE= 0.3034

PASS= 20 MEASURE= 0.3253

PASS= 21 MEASURE= 0.1170

PASS= 22 MEASURE= 0.0373

*Best results at pass 20 -
better M than for pre-
vious techniques.*

OK: .PRINTCLU

SELECT RANGE, LOWER TO HIGHER.

:

.18 21

PRINT TO TERMINAL(1) OR LINE PTR(0)?

:

.1

*Print the clustering
for a range of steps
in the clustering trace.*

*** PASS 18 ***

CLUSTER	1:	1	6	21					
CLUSTER	2:	2	3	5	7	15			
CLUSTER	3:	4	16	17	18	22			
CLUSTER	4:	8	9	10	11	12	19	20	
CLUSTER	5:	13	14						

*** PASS 19 ***

CLUSTER	1:	1	2	3	5	6	7	15	21
CLUSTER	2:	4	16	17	18	22			
CLUSTER	3:	8	9	10	11	12	19	20	
CLUSTER	4:	13	14						

*** PASS 20 ***

CLUSTER	1:	1	2	3	5	6	7	13	14	15	21
CLUSTER	2:	4	16	17	18	22					
CLUSTER	3:	8	9	10	11	12	19	20			

This clustering corresponds to the best M value.

*** PASS 21 ***

CLUSTER	1:	1	2	3	5	6	7	13	14	15	21		
CLUSTER	2:	4	8	9	10	11	12	16	17	18	19	20	22

OK: .CLUSTER

SELECT PROCEDURE - 1/2/3...

:

.4

Repeat using HIER4

OK: .READCLUS

OK: .PRINTMEA

SELECT RANGE, LOWER TO HIGHER.

:

.18 22

PRINT TO TERMINAL(1) OR LINE PTR(0)?

:

.1

PASS= 18 MEASURE= 0.0374
 PASS= 19 MEASURE= 0.1643
 PASS= 20 MEASURE= 0.2330
 PASS= 21 MEASURE= 0.1256
 PASS= 22 MEASURE= 0.0373

Best M at pass 20 - not
 as good as for HIER3

OK: .MODIFY

Try new clustering ar-
 rangement.

WHICH PASS TO BE MODIFIED?

:
.20

ENTER CHANGES, IN FORMAT: NODE NO., CLUS NO. TERMINATE WITH 0,0

:
.7 4 13 4 14 4 15 4 0 0

*** PASS 20 ***

CLUSTER	1:	1	2	3	5	6	
CLUSTER	2:	4	16	17	18	21	22
CLUSTER	3:	8	9	10	11	12	19 20
CLUSTER	4:	7	13	14	15		

Results are better than
 anything so far
 (M = 0.378)

PASS= 20 MEASURE= 0.3782
 WANT TO SAVE? (YES=1, NO=0)

:
.0

Could save this clustering
 on disk if so desired.

OK: .INTERCH

Enter the interchange algorithm
 control routine.

INCH: .INITIAL

Initialize to standard single
 subgraph

SELECT NORMAL INIT. (1) OR OWN PARTITIONING (0)

:
.1

ECHO OF INITIAL PARTITION:

# 1:	1	2	3	4	5	6	7	8	9	10	11	12	13
	14	15	16	17	18	19	20	21	22				

INCH: .SPLIT

Partition the starting
 subgraph

ENTER SUBGRAPH ID NUMBER.

:
.1

CARDINALITY OF SUBGRAPH NUMBER 1 = 22
ENTER MINIMUM SUBGRAPH SIZE.

:
.1
..

Resulting partition

INCH: .TYPEPRO

# 1:	1	2	3	5	6	7	13	14	15										
# 2:	4	8	9	10	11	12	16	17	18	19	20	21	22						

INCH: .EVALPRO

M FOR PROPOSED DECOMPOSITION = 0.133 M value for this partition.

INCH: .UPDATECU

Set the current partition to this proposed one.

CURRENT PARTITION <- PROPOSED PARTITION.

INCH: .CALCSTR

Calculate the strengths for both subgraphs in current partition.

ENTER SUBGRAPH ID NUMBER. ENTER 0 FOR "ALL".

:
.C
..

STRENGTH FOR SUBGRAPH 1 = 0.073

STRENGTH FOR SUBGRAPH 2 = 0.068 Subgraph 2 lower - select it for next partitioning.

INCH: .SPLIT

ENTER SUBGRAPH ID NUMBER.

:
.2
..

CARDINALITY OF SUBGRAPH NUMBER 2 = 13
ENTER MINIMUM SUBGRAPH SIZE.

:
.1
..

INCH: .TYPEPRO

# 1:	1	2	3	5	6	7	13	14	15										
# 2:	4	16	17	18	21	22													
# 3:	8	9	10	11	12	19	20												

Result after partitioning subgraph 2.

INCH: .EVALPRO

M FOR PROPOSED DECOMPOSITION = 0.299 M getting better

INCH: .UPDATECU

Set current partition to this proposed one.

CURRENT PARTITION <- PROPOSED PARTITION.

INCH: .CALCSTR

ENTER SUBGRAPH ID NUMBER. ENTER 0 FOR "ALL".

:
.1

STRENGTH FOR SUBGRAPH 1 = 0.073

INCH: .CALCSTR

ENTER SUBGRAPH ID NUMBER. ENTER 0 FOR "ALL".

:
.0

STRENGTH FOR SUBGRAPH 1 = 0.073

*Calculate strengths for
the three subgraphs.*

STRENGTH FOR SUBGRAPH 2 = 0.177

STRENGTH FOR SUBGRAPH 3 = 0.097

INCH: .SPLIT

ENTER SUBGRAPH ID NUMBER.

:
.1

*Now partition subgraph 1
(lowest strength)*

CARDINALITY OF SUBGRAPH NUMBER 1 = 9
ENTER MINIMUM SUBGRAPH SIZE.

:
.4

INCH: .TYPEPRO

1: 1 2 3 5 6
2: 4 16 17 18 21 22
3: 8 9 10 11 12 19 20
4: 7 13 14 15

*Resulting proposed par-
tition.*

INCH: .EVALPRO

M FOR PROPOSED DECOMPOSITION = 0.378

*Best M so far of all
approaches tried.*

INCH: .UPDATECU

CURRENT PARTITION <- PROPOSED PARTITION.

INCH: .EVALCUR

Evaluate current
partition.

M FOR CURRENT DECOMPOSITION = 0.378

INCH: .CALCSTR

ENTER SUBGRAPH ID NUMBER. ENTER 0 FOR "ALL".

:
.0

STRENGTH FOR SUBGRAPH 1 = 0.163

Repeat procedure one
more cycle.

STRENGTH FOR SUBGRAPH 2 = 0.177

STRENGTH FOR SUBGRAPH 3 = 0.097

STRENGTH FOR SUBGRAPH 4 = 0.067

INCH: .SPLIT

ENTER SUBGRAPH ID NUMBER.

:
.3

CARDINALITY OF SUBGRAPH NUMBER 3 = 7
ENTER MINIMUM SUBGRAPH SIZE.

:
.3

INCH: .TYPEPRO

# 1:	1	2	3	5	6	
# 2:	4	16	17	18	21	22
# 3:	9	10	11	19		
# 4:	7	13	14	15		
# 5:	8	12	20			

Proposed partition.

INCH: .EVALPRO

M FOR PROPOSED DECOMPOSITION = 0.041

M way down from pre-
vious partitioning -
don't accept.

INCH: .INITIAL

Try setting up own partition.

SELECT NORMAL INIT. (1) OR OWN PARTITIONING (0)

:
.0

ENTER LIST OF NODES FOR EACH CLUSTER IN TURN.
TERMINATE EACH LIST WITH A ZERO.
TERMINATE ENTIRE ENTRY WITH ANOTHER ZERO.

:
.1 2 3 5 6 9 21 0

:
.7 13 14 15 0

:
.4 16 17 18 22 0

:
.8 10 11 12 19 20 0 0

ECHO OF INITIAL PARTITION:

# 1:	1	2	3	5	6	9	21
# 2:	7	13	14	15			
# 3:	4	16	17	18	22		
# 4:	8	10	11	12	19	20	

INCH: .EVALCUR

M FOR CURRENT DECOMPOSITION = 0.371

Good results, but not
as good as earlier.

INCH: .AUTO

execute automatic governor

ENTER MINIMUM SUBGRAPH SIZE.

:
.4

BEGIN AUTO. DECOMPOSITION. USE ATTN TO STOP EARLY.

M FOR FULL GRAPH = 0.037

NEXT PARTITION IS:

1: 4 8 9 10 11 12 16 17 18 19 20 21 22
2: 1 2 3 5 6 7 13 14 15

M FOR THIS DECOMPOSITION = 0.133

result of first partitioning

SUBGRAPH STRENGTHS:

NO	STREN	CARD	MINCARD
1	0.068	13	8
2	0.073	9	8
SELECT NO.		1	

*selects subgraph 1 for
next partitioning*

NEXT PARTITION IS:

1: 8 9 10 11 12 19 20
2: 1 2 3 5 6 7 13 14 15
3: 4 16 17 18 21 22

M FOR THIS DECOMPOSITION = 0.299

SUBGRAPH STRENGTHS:

NO	STREN	CARD	MINCARD
1	0.097	7	8
2	0.073	9	8
3	0.177	6	8
SELECT NO.		2	

*note subgraph 3 is ineligible
for next split - only has
6 nodes (nmin = 4)*

NEXT PARTITION IS:

1: 8 9 10 11 12 19 20
2: 1 2 3 5 6
3: 4 16 17 18 21 22
4: 7 13 14 15

M FOR THIS DECOMPOSITION = 0.378

**** BEST M ***
same end result as found
earlier.*

SUBGRAPH STRENGTHS:

NO	STREN	CARD	MINCARD
1	0.097	7	8
2	0.163	5	8
3	0.177	6	8
4	0.067	4	8

*no subgraphs eligible now.
AUTO stops when all sub-
graphs contain fewer than
2*nmin nodes.*

FOUND NO MORE SUBGRAPHS FOR PARTITIONING.
RETURN TO INCHCTL COMMAND LEVEL.

INCH:.RETURN

return to MASTER

INTERCHANGE ANALYSIS ENDED.

stop run.

OK:.STOP

stop run.

RUN ENDED.
R;

Appendix F

EXECUTION TRACE OF INTERCHANGE ALGORITHM.

This appendix contains a computer-produced trace of the execution path calculations performed by the master control procedure and the imbedded interchange algorithm in the course of decomposing the 10-node graph first introduced in Section 6.2.3. The actual partitions effected by the algorithm as detailed here are also shown in Figure 6.15.

GRAPH ADJACENCY MATRIX

	1	2	3	4	5	6	7	8
1	0.00							
2	0.50	0.00						
3	0.80	0.40	0.00					
4	0.00	0.60	0.50	0.00				
5	0.00	0.00	0.00	0.40	0.00			
6	0.00	0.00	0.00	0.00	0.80	0.00		
7	0.00	0.00	0.00	0.00	0.40	0.90	0.00	
8	0.00	0.00	0.00	0.70	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50

(COMMENTS)

Start with entire graph

BEGIN PARTITIONING.

CLUSTERS ARE: (1 2 3 4 5 6 7 8 9 10)

MEASURE = 0.051

SET STARTING PARTITION 1

INITIAL PARTITION IS:

#1: 1 2 3 4 5 6 7 8 9

#2: 10 11 12 13 14 15 16 17 18

Add 8 dummy nodes
($n_1 = 1$)

Starting partitions chosen arbitrarily.

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	9	10	-0.100	-0.100
2	8	11	-0.600	-0.700
3	4	12	-0.800	-1.500
4	2	13	-0.300	-1.800
5	3	14	0.100	-1.700
6	1	15	1.300	-0.400
7	5	16	-0.800	-1.200
8	6	17	-0.100	-1.300
9	7	18	1.300	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 9

First starting partition leads to no interchanges.

FINAL PARTITION:

#1: 1 2 3 4 5 6 7 8 9

#2: 10 11 12 13 14 15 16 17 18

SET STARTING PARTITION 2

INITIAL PARTITION IS:

#1: 1 3 5 7 9 11 13 15 17

#2: 2 4 6 8 10 12 14 16 18

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	9	6	2.600	2.600

2	11	2	0.300	2.900
3	13	4	0.800	3.700
4	15	12	0.000	3.700
5	17	14	0.000	3.700
6	1	16	-1.300	2.400
7	3	18	-0.100	2.300
8	7	8	-1.700	0.600
9	5	10	-0.600	0.000

PARTIALMAX= 3.700 PARTIALMAX INDEX= 3

FINAL PARTITION:

#1:	1	2	3	4	5	6	7	15	17
#2:	8	9	10	11	12	13	14	16	18

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	15	11	0.000	0.000
2	17	12	0.000	0.000
3	4	13	-0.800	-0.800
4	2	14	-0.300	-1.100
5	3	16	0.100	-1.000
6	1	18	1.300	0.300
7	5	10	-1.600	-1.300
8	6	9	-0.400	-1.700
9	7	8	1.700	0.000

PARTIALMAX= 0.300 PARTIALMAX INDEX= 6

FINAL PARTITION:

#1:	5	6	7	11	12	13	14	16	18
#2:	1	2	3	4	8	9	10	15	17

BEGIN ITERATION 3

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	11	15	0.000	0.000
2	12	17	0.000	0.000
3	13	10	-0.800	-0.800
4	14	9	-0.300	-1.100
5	16	8	0.400	-0.700
6	18	4	0.000	-0.700
7	7	2	-1.600	-2.300
8	6	3	0.200	-2.100
9	5	1	2.100	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1:	5	6	7	11	12	13	14	16	18
#2:	1	2	3	4	8	9	10	15	17

Second starting partition
locates good final partition.

SET STARTING PARTITION 3

INITIAL PARTITION IS:

#1:	5	6	7	8	9	10	11	12	13
#2:	1	2	3	4	14	15	16	17	18

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	11	4	0.000	0.000
2	12	14	0.000	0.000
3	13	15	0.000	0.000
4	10	16	-0.800	-0.800
5	9	17	-0.300	-1.100
6	8	18	0.400	-0.700
7	7	2	-1.600	-2.300
8	6	3	0.200	-2.100
9	5	1	2.100	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 5 6 7 8 9 10 11 12 13
#2: 1 2 3 4 14 15 16 17 18

Same result as for
second starting par-
tition.

RESULTING PARTITIONS NOT EQUIVALENT;
BEST PARTITION IS:

#1: 5 6 7
#2: 1 2 3 4 8 9 10

"Best" is determined
via objective fun-
ction M.

END OF PASS 1

Decomposition after 1 pass.

CLUSTERS ARE: (5 6 7) (1 2 3 4 8 9 10)

MEASURE = 0.292

LOCATE SUBGRAPH WITH LOWEST STRENGTH:

SUBGRAPH NO. STRENGTH

1 0.2332

2 0.0777

Subgraph 2 is best
candidate for next
pass.

FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH: 2

SET STARTING PARTITION 1

INITIAL PARTITION IS:

#1: 1 2 3 4 5 6
#2: 7 8 9 10 11 12

Note that internal
number identi-
fiers are assigned
to the nodes of
the subgraph being
partitioned.

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	6	7	-0.100	-0.100
2	5	8	-0.600	-0.700
3	4	9	-0.400	-1.100
4	2	10	-0.300	-1.400
5	3	11	0.100	-1.300
6	1	12	1.300	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 6

FINAL PARTITION:

#1: 1 2 3 4 5 6
#2: 7 8 9 10 11 12

SET STARTING PARTITION 2

INITIAL PARTITION IS:

#1: 1 3 5 7 9 11
#2: 2 4 6 8 10 12

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	5	2	1.100	1.100
2	7	4	1.200	2.300
3	9	8	0.000	2.300
4	11	10	0.000	2.300
5	1	12	-1.300	1.000
6	3	6	-1.000	0.000

PARTIALMAX= 2.300 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 1 2 3 4 9 11
#2: 5 6 7 8 10 12

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	9	8	0.000	0.000
2	11	10	0.000	0.000
3	4	12	-0.400	-0.400
4	2	7	-1.100	-1.500
5	3	6	-0.200	-1.700
6	1	5	1.700	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 1 2 3 4 9 11
#2: 5 6 7 8 10 12

SET STARTING PARTITION 3

INITIAL PARTITION IS:

#1: 4 5 6 7 8 9
#2: 1 2 3 10 11 12

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	4	10	0.400	0.400
2	8	11	0.000	0.400
3	9	12	0.000	0.400
4	5	1	-1.700	-1.300
5	6	3	0.200	-1.100
6	7	2	1.100	0.000

PARTIALMAX= 0.400 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 5 6 7 8 9 10
#2: 1 2 3 4 11 12

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	8	11	0.000	0.000
2	9	12	0.000	0.000
3	10	4	-0.400	-0.400
4	7	2	-1.100	-1.500
5	6	3	-0.200	-1.700
6	5	1	1.700	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 5 6 7 8 9 10
#2: 1 2 3 4 11 12

RESULTING PARTITIONS NOT EQUIVALENT;

BEST PARTITION IS:

#1: 1 2 3 4
#2: 5 6 7

*Result after second
pass (turns out to
be optimal result)*

END OF PASS 2

CLUSTERS ARE: (5 6 7) (1 2 3 4) (8 9 10)

MEASURE = 0.484

LOCATE SUBGRAPH WITH LOWEST STRENGTH:

*These are "real" node
numbers (not in-
ternal identifiers).*

SUBGRAPH NO. STRENGTH

1	0.2332
2	0.1866
3	0.1555

FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH: 3

SET STARTING PARTITION 1

INITIAL PARTITION IS:

#1: 1 2
#2: 3 4

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	4	-0.100	-0.100
2	2	3	0.100	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 1 2
#2: 3 4

SET STARTING PARTITION 2

INITIAL PARTITION IS:

#1: 1 3
#2: 2 4

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	4	0.100	0.100
2	3	2	-0.100	0.000

PARTIALMAX= 0.100 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 3 4
#2: 1 2

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	3	2	-0.100	-0.100
2	4	1	0.100	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 3 4
#2: 1 2

SET STARTING PARTITION 3

INITIAL PARTITION IS:

#1: 2 3
#2: 1 4

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	2	4	0.300	0.300
2	3	1	-0.300	0.000

PARTIALMAX= 0.300 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 3 4
#2: 1 2

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	3	2	-0.100	-0.100
2	4	1	0.100	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 3 4
#2: 1 2

ALL PARTITIONS EQUIVALENT, SO
BEST PARTITION IS:

#1: 1 2
#2: 3

END OF PASS 3
 CLUSTERS ARE: (5 6 7) (1 2 3 4) (8 9)
 MEASURE = -0.101

LOCATE SUBGRAPH WITH LOWEST STRENGTH:

SUBGRAPH NO.	STRENGTH
1	0.2332
2	0.1866
3	0.0000

FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH: 3 No calculations: subgraph only has 2 nodes.

END OF PASS 4
 CLUSTERS ARE: (5 6 7) (1 2 3 4) (8)
 MEASURE = -1.189

LOCATE SUBGRAPH WITH LOWEST STRENGTH:

SUBGRAPH NO.	STRENGTH
1	0.2332
2	0.1866

Note that strengths are only calculated for subgraphs with

FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH: 2 at least 2 nodes.

SET STARTING PARTITION 1

INITIAL PARTITION IS:

#1: 1 2 3
 #2: 4 5 6

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	4	-0.200	-0.200
2	3	5	-0.100	-0.300
3	2	6	0.300	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 3

FINAL PARTITION:

#1: 1 2 3
 #2: 4 5 6

SET STARTING PARTITION 2

INITIAL PARTITION IS:

#1: 1 3 5
 #2: 2 4 6

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	5	2	0.300	0.300
2	1	4	-0.200	0.100
3	3	6	-0.100	0.000

PARTIALMAX= 0.300 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 1 2 3

#2: 4 5 6

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	4	-0.200	-0.200
2	3	5	-0.100	-0.300
3	2	6	0.300	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 3

FINAL PARTITION:

#1: 1 2 3

#2: 4 5 6

SET STARTING PARTITION 3

INITIAL PARTITION IS:

#1: 2 3 4

#2: 1 5 6

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	4	1	0.200	0.200
2	2	5	-0.300	-0.100
3	3	6	0.100	0.000

PARTIALMAX= 0.200 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 1 2 3

#2: 4 5 6

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	4	-0.200	-0.200
2	3	5	-0.100	-0.300
3	2	6	0.300	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 3

FINAL PARTITION:

#1: 1 2 3

#2: 4 5 6

ALL PARTITIONS EQUIVALENT, SO

BEST PARTITION IS:

#1: 1 2 3

#2: 4

END OF PASS 5

CLUSTERS ARE: (5 6 7) (1 2 3) (8) (10) (9)(4)

MEASURE = -2.178

LOCATE SUBGRAPH WITH LOWEST STRENGTH:

SUBGRAPH NO. STRENGTH

1 0.2332

2 0.1888

FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH: 2

SET STARTING PARTITION 1

INITIAL PARTITION IS:

#1: 1 2

#2: 3 4

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	4	0.300	0.300
2	2	3	-0.300	0.000

PARTIALMAX= 0.300 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 2 4

#2: 1 3

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	2	3	-0.300	-0.300
2	4	1	0.300	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 2 4

#2: 1 3

SET STARTING PARTITION 2

INITIAL PARTITION IS:

#1: 1 3

#2: 2 4

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	4	-0.300	-0.300
2	3	2	0.300	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 1 3

#2: 2 4

SET STARTING PARTITION 3

INITIAL PARTITION IS:

#1: 2 3

#2: 1 4

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	2	1	0.400	0.400
2	3	4	-0.400	0.000

PARTIALMAX= 0.400 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 1 3
#2: 2 4

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	4	-0.300	-0.300
2	3	2	0.300	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 1 3
#2: 2 4

ALL PARTITIONS EQUIVALENT, SO
BEST PARTITION IS:

#1: 2
#2: 1 3

END OF PASS 6

CLUSTERS ARE: (5 6 7) (2) (8) (10) (9) (4)
MEASURE = -3.300

LOCATE SUBGRAPH WITH LOWEST STRENGTH:

SUBGRAPH NO.	STRENGTH
1	0.2332
7	0.0000

FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH: 7

END OF PASS 7

CLUSTERS ARE: (5 6 7) (2) (8) (10) (9) (4)
MEASURE = -4.800

LOCATE SUBGRAPH WITH LOWEST STRENGTH:

SUBGRAPH NO.	STRENGTH
1	0.2332

FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH: 1

SET STARTING PARTITION 1

INITIAL PARTITION IS:

#1: 1 2
#2: 3 4

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	3	0.100	0.100
2	2	4	-0.100	0.000

PARTIALMAX= 0.100 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 2 3
#2: 1 4

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	2	4	-0.100	-0.100
2	3	1	0.100	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 2 3
#2: 1 4

SET STARTING PARTITION 2

INITIAL PARTITION IS:

#1: 1 3
#2: 2 4

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	1	2	0.500	0.500
2	3	4	-0.500	0.000

PARTIALMAX= 0.500 PARTIALMAX INDEX= 1

FINAL PARTITION:

#1: 2 3
#2: 1 4

BEGIN ITERATION 2

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	2	4	-0.100	-0.100
2	3	1	0.100	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 2 3
#2: 1 4

SET STARTING PARTITION 3

INITIAL PARTITION IS:

#1: 2 3
#2: 1 4

BEGIN ITERATION 1

INTERCHANGE TRACE:

CYCLE	NODE 1	NODE 2	MAX GAIN	PAR SUM
1	2	4	-0.100	-0.100
2	3	1	0.100	0.000

PARTIALMAX= 0.000 PARTIALMAX INDEX= 2

FINAL PARTITION:

#1: 2 3
#2: 1 4

ALL PARTITIONS EQUIVALENT, SO

BEST PARTITION IS:

#1: 2 3
#2: 1

END OF PASS 8

CLUSTERS ARE: (6 7) (2) (8) (10) (9) (4) (1) (3) (5)
MEASURE = -5.900

LOCATE SUBGRAPH WITH LOWEST STRENGTH:

SUBGRAPH NO.	STRENGTH
1	0.0000

FOR NEXT PARTITIONING ROUND, SELECT SUBGRAPH: 1

END OF PASS 9

CLUSTERS ARE: (6) (2) (8) (10) (9) (4) (1) (3) (5) (7)
MEASURE = -7.400

OVERALL BEST PARTITION:

OPTIMUM MEASURE M* = 3.4837
OPTIMAL DECOMPOSITION IS:

(5 6 7)
(1 2 3 4)
(8 9 10)

Overall optimal
partition re-
membered and
printed out.

MONTHLY ANALYSIS

SUPERVISOR D. T. OTHERWAY
ROOM E40-250ADDRESSEE F. M. FINK
ROOM E40-211PAGE 1 AS OF DATE
OF 2 02-28-77DEPARTMENT NAME
CENTER FOR ALTERNATE LIFESTYLEDEPT. NO.
046900EFFECTIVE
10-01-76EXPIRES
09-30-78ACCOUNT NAME
CELESTIAL NAVIGATION OF TADPOLESACCT. NO.
04559

SPONSOR NATIONAL PROG FOUNDATION

REFERENCE 7682054-PRA

PARENT ACCT 04545

OBJ. DESCRIPTION AND NOTES CODE	PRO- RATE CODE	ACTUAL CURRENT MONTH	FISCAL YTD	REMAINING F.Y. BUDGET F.Y. TOTAL (FYTD)-(REH)	F.Y. BUDGET AUTHORIZED 02-11-77	C M G	FISCAL YEAR VARIANCE (BUD)-(PROJ)	M O T E	CUMULATIVE AMOUNT	TOTAL AUTHORIZED
EXPENDITURES										
SALARIES AND WAGES	E.F.T.									
214 OTHER ACAD. STAFF-ON	2.3 12	1.154	1.904	3.400	5.504		10.000		1.904	5.402
320 SECRETARIAL & CLERK-ON	-0 12			1.072	1.072		5.416		3.744	7.408
346 TECH OR ADMIN SUPT-ON	-0 12		%	934	1.010	1	2.001		%	5.402
900 TOTAL SUBJ TO E.B.	2.3 10	1.154	2.001	6.406	6.406		19.217		2.001	51.269
350 GRAD STUDENT STAFF-ON	-0			1.295	1.295		3.453			10.340
900 TOTAL SLAT & WAGES	2.3 10	1.154	2.001	7.701	9.701		22.670		2.001	61.629
EMPLOYEE BENEFITS ON OFF										
401 BILLING RATES 24.50X 20.50X	10	203	490	1.007	2.059		4.710		490	12.561
900 TOTAL SALARIES, WAGES & E.B.	10	1.437	2.491	9.500	11.760		27.380		2.491	74.290
OPERATING EXPENSES										
415 TRAVEL	01									
419 MATERIALS AND SERVICES	12		01	1.031	1.112		2.319		01	1.000
523 XEROX EXPENSE	01		34		34				34	6.104
810 TEL-TOLL CALLS	01	19	34		34				34	
991 TOTAL OPERATING EXPENSES	10	19	149	1.031	1.100		2.319		149	7.104
COMPUTATION EXPENSES										
096 IPC CHARGES	12	140	349	167	516		375		349	1.000
099 TOTAL COMPUTATION EXPENSES	10	140	349	167	516		375		349	1.000
OTHER CHARGES										
901 OTHER CHARGES	17									3.000
839 TOTAL OTHER CHARGES	10									3.000

SAMPLE

Appendix H

INITIAL BUDGETING SYSTEM FUNCTIONAL REQUIREMENTS AS PREPARED BY THE SYSTEM DESIGNERS.

1. Automate as many manual procedures as feasible to save time and effort
2. Provide in the Chart of Accounts the following for all fund accounts:
 - a) Coding for Principal - Endowment, Quasi-endowment, or Term Endowment
 - b) Coding for Income or current funds - Unrestricted, Restricted, Designated, or Restricted and Further Designated.
 - c) Update the 60-character 'Fund Purpose' field
3. Collect and store planning data contributed by the senior managers
 - a) Data must be in reasonably uniform, useful format
 - b) Data should include short range (Next fiscal year) and long range (3 to 5 years) plans
 - c) The report of the senior managers contained in the 'Report of the President and the Chancellor' should refer to and be consistent with the short range plan presented prior to the beginning of the fiscal year.
 - d) Planning data would be available to the Dynamic Model (or its replacement)
4. Provide for collecting, storing and reviewing details of special financial agreements made among managers. Details could be picked up when drafts

are processed. These might be input at time drafts are made.

5. Provide for budgeting and reporting by E.F.T. as well as dollars. An interface with the Payroll System may be necessary.
 - a) Automate system where possible so that a person may be budgeted either by E.F.T. or dollars and provide a method for one to generate the other
 - b) Allow for either actual or average salaries to be used in budget proposals.
 - c) Provide checks to ensure that each person is not budgeted more than 100% E.F.T.
6. Provide for producing reports in units of man-weeks, man-months or man-years
7. Provide for additional object codes to be summaries of other object codes as necessary
 - a) Use summary codes for subtotals on reports or for reporting only these subtotals
 - b) Explore usefulness of summary codes that would correspond with Billing Codes
8. Review Object Code descriptions to ascertain their current applicability
9. Provide support for special object codes for use by each department. The descriptions for these codes would be supplied by the using departments, and they would subtotal to the appropriate summary object codes.
10. Reports must be easier to handle and store. Both the IBM 3800 and Xerox 9700 Printers would solve this problem through the use of 8 1/2" by 11" size paper without sacrificing data capacity.
11. Provide for sharing budget data with users of other systems and for use of the files of other systems by the Budget System (e.g. Gift System, AFS, Payroll, Purchasing, etc.)
12. Provide for as much early recognition of potential problems as possible so that corrections can be made in time to prevent them

- a) Provide easy way to project income and expenses over the fiscal year on a month by month basis so as to provide a ready measure of actual performance vs. budget.
- b) Provide way to flag amounts or items when pre-determined tolerances or dates are exceeded

13. Provide and maintain access to data:

- a) Chart of Accounts
- b) Future budget data by month by object code within account
- c) Current fiscal year budget data by month by object code within account
- d) Last fiscal year budget data by month by object code within account
- e) Current fiscal year actual data by month by object code within account
- f) Last fiscal year actual data by month by object code within account
- g) Historical data, up to 10 years of budget and actual data by object summary within account
- h) Salary and other data required for proposal preparation

14. Promote optimal use of fund accounts so as to conserve general monies

- a) Provide directory (or directories) and an application index of funds so as to readily show how they can be applied. This would require collection and maintenance of abbreviated text of fund description and donor's intentions.
- b) Enhance existing reports (X52, X53 and/or X56) with available data so as to make them more useful in applying funds.
- c) Provide for production of a report in X52/X53 format showing 10 years of historical data.

- d) Explore other ways to promote use of funds when it is optional, such as matching fund expenditures with a certain amount of General money.
15. Support monitoring of operating budget
- a) Budget vs. actual
 - b) 'Operating gap'
16. Support, improve or replace the following reports or items of information:
- a) Printed Budget
 - i) In detail by account within departments and areas of responsibility
 - ii) In 'Schedule A' format for inclusion in the Treasurer's Report
 - iii) In functional format
 - b) Schedules for the Treasurer's Report
 - c) Indirect Cost Recovery Percentage based on CAC Studies
 - d) Analyze effects of potential 'dollar budgeting' decisions, such as 1% salary change, etc. Be sure to provide for changing scholarship, fellowship and stipend rates when changing tuition.
 - e) MITOP
 - f) Dynamic Model
 - i) Provide for automatic interrelationship so that when data changes the subsequent effects are shown
 - ii) Provide for ease of rerunning with changes in variables and assumptions to answer 'what if' questions
 - iii) Explore use of Boeing's Executive Information System or other packages to support modeling

- q) Periodic Summary of Operations
- h) Budget Authorizations and Changes, and explanations of the changes, including dollar budgeting.
 - i) Provisional and approved Budget Authorizations
 - ii) Changes in authorized amounts or allocations and explanations of same, including dollar budgeting
- i) Gross and net effect of budget changes, including, but identifying, 'dollar budgeting'
- j) Budget vs. Actual Reports (X80, X83, X84)
- k) Detailed Transaction Report
 - i) Print account title on DTR
 - ii) Include detail of Purchase Order Commitments
 - iii) Provide more information in description and reference fields so as to facilitate tracking expenses
 - iv) Expand data content to include YTD and cumulative data
 - v) Do not produce DTR in months when there is no activity for an account
- l) Monthly Statement
 - i) Consider showing 'Travel Outstanding' immediately following 'Travel' and subtotalling them
 - ii) Supply meaningful Purchase Order Commitment data
- m) List of accounts about to expire
- n) Budget proposal forms and supporting documents
- o) Area of Responsibility Report

- p) Department Profiles
 - q) Report of Fund Drafts
 - r) Physical Plant Summary
17. Supply new reports as required for fiscal planning and budgeting
- a) Analysis of faculty support, in both E.F.T. and dollars, contributed by laboratories to academic departments
 - b) Report on research type, discipline and function so as to satisfy both NSF and MIT requirements
 - c) Monthly Analysis
 - i) Determine if Monthly Analysis should be published in place of or in addition to Monthly Statement
 - ii) Investigate rounding of dollar amounts to nearest 1
 - iii) Investigate feasibility of not publishing Monthly Analysis for months in which there is no activity in an account
 - iv) Provide for including or excluding certain data depending on user requirements, i.e. eliminate billing and fiscal year-to-date data from report to principal investigator, etc.
 - v) In forms design try to show subsections by rounding tops of column heads
 - vi) Consider showing 'Travel Outstanding' immediately following 'Travel', and sub-totaling them
 - vii) Print account numbers in lower right corner for easy lookup when reports are filed in a notebook
 - viii) Print notes to show special restrictions.

- ix) Provide for flexibility in format for content (optional columns)
 - x) Print notes that describe budget changes
 - xi) Supply meaningful Purchase Order Commitment data
 - d) Summary Analysis for:
 - i) Parent Accounts
 - ii) Departments and subdepartments
 - iii) Schools
 - iv) Areas of Responsibility
 - v) The Institute
 - vi) Principal Investigator
 - vii) Research Type, Discipline and/or function
 - viii) Other subdivisions as required
 - e) Terminal formats for entering budget proposals
 - f) Budget Projection - Month by month projection by object codes for one or a group of accounts.
 - g) Additional reports used by certain departments, such as K. Keays series
18. On Personnel Action Form add a box to indicate whether person hired is a replacement or an addition.
19. Support Special reports for budget-related activities
- a) Standard reports at non-standard times
 - b) Standard reports for non-standard periods
 - i) Contract period
 - ii) Sponsor's fiscal year
 - c) Standard data in non-standard formats

- d) Report-writer language for fully customized reports. This language must be easily learned and used.
20. Support for on-line operations with budget data-base
- a) 'Menu' for standard inquiry series
 - b) Inquiry language for special use
 - c) Support for on-line requests for reports to be printed either on terminal or at remote printer
21. Provide data security and recoverability
- a) Protect sensitive data against unauthorized use by checking and logging identity of user and, possibly, by encoding data
 - b) Prevent data from being changed by unauthorized persons, access would be 'read only' except to its owner
 - c) System must provide data integrity through logging of changes and capability of reversing them
 - d) Provide adequate file backup procedures
22. Provide a simplified proposal preparation procedure that will support both on-line and manual preparation of budgets.
23. Investigate proposal preparation using primarily summary object codes.
- a) Standard MIT monthly statement summary object codes for subtotals
 - b) Billing summary object codes
24. Provide for review of budget proposals and changes by the Budget Office in simplified ways
- a) Investigate feasibility of manually-prepared machine-readable budget proposal worksheets
 - b) Provide for review and acceptance of proposals prepared either manually or on-line

- c) Provide maximum effective amount of computer editing of budget proposals
 - d) Provide check to be sure proposals are within target amount
 - e) Compare actual E.F.T. and head count against allowances
 - f) Provide check to be sure every open account has a proposal
 - g) Be able to tell status of every proposal
- 25. Identify 'base general' in budget
 - 26. Provide for changing Employee Benefit amounts when changes are made affecting the amount of Salary and Wages budgeted
 - 27. Encourage the use of program budgeting while still supporting other effective budgeting techniques currently in use
 - 28. Provide logical integrity of budget data by assigning ownership and responsibility for each data element. Administrators should be able to change certain elements or allocations on their own authority, whereas others should require Budget Office approval
 - 29. Provide a system for tracking research proposals so as to show whether they are accepted or rejected, what amount of funding is made available, funding agency, principal investigator, length of contract, special restrictions, and other appropriate information. Record OSP proposal number on 001 form.
 - 30. Support the establishment and use of a system of Discipline/Function codes for research projects
 - 31. Eliminate the need for certain departmentally-produced reports by making available either standard or special reports.
 - 32. Promote simplification of the budgeting and financial planning operations

33. Provide documentation and audit trail required for follow-up of discrepancies in system, operational or user areas.
34. Support handling of all required aspects of special items such as:
 - a) Nonrecurring equipment
 - b) Carryforward amounts
 - c) Sabbatical leaves
 - d) Drafts
 - e) Reserves
 - f) Other special requests
35. Support Fund Drafts by computer via on-line and batch as well as manual (log sheet) requests from user departments. Use on-line checking by Budget Office where feasible.
36. Provide for recording joint or non-standard support for an account
37. Determine desirability and feasibility of encumbering salary and wage budgeted amounts.
38. Establish data base to support budget activities as well as A&S and other data or systems
39. Investigate the feasibility of updating actual income and expense data more frequently than once a month.
40. Review system of fund purpose codes to see if they could be made more beneficial to both gift system and fund users.
41. Provide for communication between Budget System and Gift System.
42. Provide for additional data and types of data to be stored as requirements occur
 - a) Provide for an open-ended system of descriptors for accounts or objects within an account to carry whatever supplementary data may be needed by users. This would include:

- i) Data on special funding and non-recurring expenses
 - ii) Indicators as to Functional Summary categories
 - iii) Complete set of applicable Donor Purpose codes for fund accounts
 - b) Provide for access to data via the special descriptors
43. Provide for discussion of all projected changes with affected users whenever feasible
44. Provide formal training for users when system is introduced and periodically thereafter
45. Provide, maintain and distribute adequate documentation for all system users.
46. Support various Employee Benefit Codes and Overhead Recovery Rates as appropriate
- a) Ensure that data is available to show employee benefit and overhead recovery rates appropriate to the accounting period.
 - b) Support individual employee benefit rates to accommodate personnel from other universities who work on projects at M.I.T., such as the consortium for the Center for Materials Research in Archaeology and Technology
47. Speed up delivery of reports to users
48. Explore possibility of accounting cutoff at the end of the month

Appendix I

FINAL BUDGETING SYSTEM FUNCTIONAL REQUIREMENTS AS USED IN THE SDM ANALYSIS.

1. Fund accounts will be categorized by fund purposes.
2. Fund accounts can be categorized by principal type.
3. Fund accounts can be categorized by income type.

COMMENT:

Principal types may include endowment, quasi-endowment, and term endowment.

Income types may include: Unrestricted, Restricted, Designated, or Restricted and Further Designated.

4. Each fund account will be described via abbreviated text description.

COMMENT:

The current 60-character fields may be used for this purpose.

5. Short-term and long-term planning data will be provided by Senior Managers.

COMMENT:

Senior Managers are: Deans, Department Heads, Lab Directors, and Vice Presidents.

6. Planning data will be provided in a standardized format.

COMMENT:

Department profile format.

7. Data regarding special financial arrangements made by managers will be maintained.

COMMENT:

Could be between managers or between fund donors and managers.

8. Personnel budgets can be developed in dollars.

COMMENT:

Either average or actual figures.

9. Personnel budgets can be developed in EFT.

10. There will be a facility for converting salary/wage information between dollars and EFT.

COMMENT:

This conversion should be as automatic as possible.

11. Certain objects may be logically related to groups of other objects.

COMMENT:

For example, summary object codes.

12. A particular object may belong to multiple object groups.

COMMENT:

For example, certain codes may be reserved as summary codes for specific groups of other object codes. Summary codes might include subtotal information.

13. Manpower can be reported in alternative units.

COMMENT:

For instance, dollars, man-days, man-months, man-years.

14. There will be special object codes available for use by each department.

COMMENT:

Each department would determine its own categorizations.

Instance: additional subdivisions on travel:
Travel to LA, Travel to Washington, etc.

15. Reports will be physically easy to handle.

COMMENT:

3800 printer would help to do this by increasing data capacity per page.

16. Budgeting and planning data can be accessed readily by other systems.

17. The budget system can access directly data in other systems' files.

COMMENT:

Including ARS, Payroll, Gift systems.

18. There will be a general comparison reporting capability.

COMMENT:

May be budget-versus-budget or budget-versus-actual.

May be for different time periods (e.g., a particular week this year versus same week last year).

Other examples would be: one department versus rest of school; comparisons between budgets of different principal investigators.

19. Items exceeding prespecified bounds can be highlighted.

20. Chart of Accounts and associated supplementary data will be accessible by Budget System.

21. Current fiscal year budget data will be maintained.
22. Future Budget data will be maintained.
23. Last fiscal year Budget data will be maintained.
24. Current fiscal year Actual data will be maintained.
25. Last fiscal year Actual data will be maintained.

COMMENT:

In general, income and expense data will be accessible by month and object code within account.

26. Historical Budget data will be maintained for up to 10 years.

COMMENT:

May be aggregated; will be off-line.

27. Historical Actual data will be maintained for up to 10 years.

COMMENT:

May be aggregated; will be off-line.

28. Data required for budget proposal preparation and not available elsewhere will be maintained.

COMMENT:

e.g., salary data from payroll system; target data.

29. Budgeted income and expense data will be prorated automatically on a month-by-month basis.

COMMENT:

Prorations can be selected from standard profiles.

Special proration profiles may be supplied by managers.

There will also be a "no proration" alternative.

30. Information to facilitate the effective use of fund accounts will be available.

COMMENT:

Provide directory (or directories) and an application index of funds so as to readily show how they can be applied. This would require collection and maintenance of abbreviated text of donor's intentions and fund description.

Enhance existing reports (X52, X53 and/or X56) with available data so as to make them more useful in applying funds.

Provide for production of a report in X52/X53 format showing 5 years of historical data.

Explore other ways to promote use of funds when it is optional, such as matching fund expenditures with a certain amount of General money.

31. Monitoring of the operating budget will be supported.

COMMENT:

e.g., budget-versus-actual reports; operating gap analysis.

32. Various Future Budget reports will be generated.

COMMENT:

Support, improve, or replace the following reports:

Printed budget:

In detail by account within departments and areas of responsibility.

In 'Schedule A' format for inclusion in Treasurer's report.

In functional format.

Physical plant summary.

Department profiles.

Budget Authorizations and Changes, and explanations of the changes, including dollar budgeting.

Provisional and approved Budget Authorizations.

Changes in authorized amounts or allocations and explanations of same, including dollar budgeting.

Gross and net effect of budget changes, including, but identifying, 'dollar budgeting'.

Budget proposal forms and supporting documents.

33. Periodic operating reports for each account will be generated.

COMMENT:

Reports will include current budget and current actual-to-date data.

Modified detailed transaction report (ARS).

Print account title on DTR.

Include detail of Purchase Order Commitments.

Provide additional information in description and reference fields so as to facilitate tracking of expenses.

Expand data content so as to include YTD and cumulative data.

Do not produce DTR in months when there is no activity on account.

Monthly statement or its replacement (monthly analysis).

Consider showing 'travel outstanding' immediately after 'travel,' and subtotalling them.

Supply meaningful purchase order commitment data.

Budget versus Actual reports (X80, X83, X84).

34. Various special-purpose (non-periodic) reports will be generated.

COMMENT:

Schedules for Treasurer's Report.

Indirect cost recovery percentage, based on CAO studies.

MITOP.

Periodic Summary of Operations.

List of accounts about to expire.

35. The data necessary for use by the Dynamic Model will be generated.

COMMENT:

The model will support automatic interrelationships of variables.

36. Ad hoc requests for information will be supported.

COMMENT:

For example, analyzing the effects of potential 'dollar budgeting' decisions, such as 1 percent salary changes, etc.

37. Various funds reports will be generated.

COMMENT:

In particular, the report on Funds drafts.

Also Funds Schedule for Treasurer's Report.

38. The system will have access to certain personnel hiring data.

COMMENT:

AD-A074 911

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
A SYSTEMATIC METHODOLOGY FOR DESIGNING THE ARCHITECTURE OF COMP--ETC(U)
SEP 79 S L HUFF, S E MADNICK
CISR-P010-7906-12

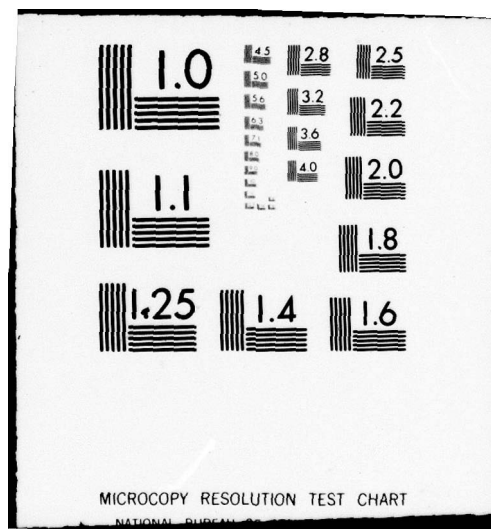
N00039-78-G-0160

NL

UNCLASSIFIED

7 OF 7
ADA
074911





Monitoring head count allowances.

Add box on Personnel Action form to indicate whether person hired is a replacement or an addition.

39. Certain budget report data items can be optionally included or excluded from Monthly Analysis Report as specified by the user.

COMMENT:

e.g., eliminate billing and fiscal YTD data from report to principal investigator.

40. Variations of standard Budget System reports will be developed as requested.

COMMENT:

Analysis of faculty support, in both E.F.T. and dollars, contributed by laboratories to academic departments.

Report on research type, discipline and function so as to satisfy both NSF and MIT requirements.

41. New budget reports will be developed as required.

COMMENT:

e.g., Monthly Analysis:

Determine if Monthly Analysis should be published in place of or in addition to Monthly Statement.

Investigate rounding of dollar amounts to nearest 1.

Investigate feasibility of not publishing Monthly Analysis for months in which there is no activity in an account.

Provide for including or excluding certain data depending on user requirements, i.e. eliminate billing and fiscal year-to-date data from report to principal investigator, etc.

In forms design try to show subsections by rounding tops of column heads.

Consider showing 'Travel Outstanding' immediately following 'Travel', and subtotaling them.

Print account numbers in lower right corner for easy lookup when reports are filed in a notebook.

Print notes to show special restrictions.

Provide for flexibility in format for content (optional columns).

Print notes that describe budget changes.

Supply meaningful Purchase Order Commitment data.

- 42. There will be a simple report-writing facility.
- 43. Users can develop their own special reports using the report writing facility.
- 44. There will be an on-line menu-oriented query facility.
- 45. There will be an on-line ad hoc query facility.
- 46. Reports generated on-line can be directed to the system printer or to users' printer/terminal.
- 47. Identity and activity of users will be logged.

COMMENT:

Should be provided by DEMS.

- 48. Data items can be accessed only by permissible users.
- 49. Data items can be changed only by their owner.
- 50. There will be a set of permissible users for each data item.
- 51. Each data item will have a unique owner.

52. Data items can be encoded for security.
53. There will be a log of all database changes.
54. There will be a file backup facility.
55. The database's integrity can be restored using backup files and change log if necessary.
56. Budget proposals can be prepared manually.
57. Budget proposals can be prepared on-line.
58. Budget proposals will be automatically checked and edited to the extent possible.

COMMENT:

Every open account must have an associated proposal.

EFT cannot exceed 100 percent per person.

59. There will be controlling limits on amounts in proposed budgets and budget changes.

COMMENT:

Check that proposals are within target amount, or within sponsor's limit.

Compare actual EFT and head count against allowances.

May be desirable to have controls on summaries also.

60. Budget proposals will be reviewed by the budget office and/or OSP.
61. Budget changes will be reviewed by the budget office and/or OSP.
62. Sources of funding for each account will be identified.

COMMENT:

Identification information may be included in budget account record.

- 63. Budgeted employee benefits will be changed automatically following changes in budgeted salary and wage amounts or EB rates.
- 64. All currently used budgeting techniques will be supported.

COMMENT:

Including line item budgeting, program budgeting, and task-oriented budgeting.

System should promote program budgeting.

- 65. Research proposal information can be stored, updated, and accessed.

COMMENT:

Provide various kinds of information for tracking research proposals to show whether they are accepted or rejected, what amount of funding is requested and made available, funding agency, principal investigator, length of contract, special restrictions, and other appropriate information. Record CSP proposal number on 001 form.

- 66. Research accounts and proposals will be categorized by type, discipline, and function.
- 67. There will be documented audit trails for determining system discrepancies.
- 68. Required aspects of non-recurring expenses will be supported.

COMMENT:

Concerns primarily sabbatical leaves, drafts, carryforward amounts, and nonrecurring equipment.

- 69. Fund drafts can be performed on-line.
- 70. Fund drafts can be performed in batch mode.
- 71. Fund drafts can be checked and approved on-line by Budget Office.

COMMENT:

Check may be done by Budget Office, or the responsible officer.

- 72. Additional data item types can be added to accounts or objects within accounts.
- 73. New data items can include or refer to supplementary data needed by users.
- 74. Formal training in the use of the system will be provided to users.

COMMENT:

Training and use documentation will be made available to legitimate system users.

- 75. Non-standard employee benefit calculations will be supported.

COMMENT:

Support individual employee benefit rates to accommodate personnel from other universities who work on projects at MIT (e.g., Consortium for Materials Research in Archeology and Technology).

- 76. Overhead recovery rate calculations will be supported.
- 77. Standard periodic reports will be produced and distributed to users as quickly as possible.

COMMENT:

Minimize time between cutoff date and report receipt.

3800 printer will help by shortening printout time and by printing in distribution sequence.

Appendix J

FORM USED IN GATHERING INTERDEPENDENCY DATA.

The data form shown on the following page was used for gathering the interdependency data during the interdependency analysis phase of the study.

INTERDEPENDENCY ASSESSMENT

SYSTEM:

[illegible]

Appendix K

BUDGETING SYSTEM REQUIREMENT INTERDEPENDENCIES.

FR	TO	WT	DESCRIPTION
1	4	S	Text description will reinforce and describe the categorization.
1	17	A	Fund purpose data may be stored elsewhere.
1	20	A	Fund purpose data may also be in Chart of Accounts.
1	30	S	Fund purpose code is a important instance of "effective-use" information.
1	36	W	Possible "fund purpose" report.
1	37	W	Possibly want to include fund purpose information.
1	58	A	Check budget proposals regarding fund purpose.
1	60	A	Reviewer would check fund purpose for consistency.
1	71	W	Can be checked for consistency.
1	72	S	May want to add new purposes.
2	4	S	Text description will reinforce and

describe the categorization.

2 17 W Fund purpose data may be stored elsewhere.

2 20 A Chart of Accounts contains fund accounts.

2 30 S Principal type is critical effective use
information.

2 36 W Possible fund report.

2 37 A Information to be included in fund reports.

3 4 S Text description will describe and
reinforce categorization.

3 7 A Often have special income restrictions.

3 17 W Gift system is source for fund account
data.

3 19 A Accumulated income important to monitor.

3 20 A Carried in Chart of Accounts.

3 30 S Income type important regarding how spent.

3 36 A Likely fund report.

3 37 S Possibly want to include fund purpose
information here.

4 17 A Other systems may need to access descrip-
tion data.

4 30 S Fund purpose description used in directory
entry.

4 36 W Text description likely target of ad hoc
requests.

4 37 S Text description would be included in many
reports.

5 6 S Use and capture of planning data related to
format.

5 22 S Related information sources.

5 34 A Might want to develop projection reports.

5 35 S Major purpose of planning data is to
provide DM input.

6 22 W Common formatting issue.

6 34 W Summary planning reports possibly contin-
gent on planning data format.

6 35 A Format affects what can be requested.

7 30 A Special arrangements for funds.

7 37 W May want to report information regarding
special arrangements.

7 58 A May be able to automate checking to include
SFA.

7 59 A May be limits or controls on SFAs.

7 60 W SFAs are grounds for examination.

7 62 S Routine examinations.

7 68 S An SFA is a special item of importance.

7 71 W SFAs would need to be checked.

8 10 S Common conversion issue.

8 13 A Common conversion issues.

8 29 A Proration of personnel expenses important
issue.

8 63 S Employee benefits developed from personnel
budgets in dollars.

8 75 S Employee benefit calculations need personnel
information in dollars.

9 10 S Common conversion issue.

9 13 S EFT required for certain reporting units.

9 29 A Proration of personnel expenses important
issue here.

9 33 A Standard reports include manpower.

9 38 W Need EFT to properly monitor headcounts.

9 59 W One control item.

10 13 S Common conversion issues.

10 28 A Budget proposal data may need conversion
facilities.

10 38 W May need conversion facilities to make
effective use.

11 12 S Common logical relationships issue.

11 14 A Special codes have to summarize correctly.

11 34 A Logical relationships used in summarizing.

11 35 A Model uses mostly summary data, and logical
relationships define certain summarizations.

12 14 A Department summary codes need logical
groupings.

12 34 S Summarizations defined in logical groupings.

15 77 S Common preparation issue - easier handling
makes for faster delivery.

16 48 A Allowable accesses must be controlled for
other systems as well as individuals.

16 49 A Need change protection as regards other
systems.

16 52 W Other systems may need encryption key access.

17 20 S Budget system will need access to Chart data.

17 69 W Need link to Funds system.

17 71 W Need Funds system link.

18 19 S Comparisons indicate items to be highlighted.

18 22 A Data used in comparisons.

18 23 A Data used in comparisons.

18 24 A Data used in comparisons.

18 25 A Data used in comparisons.

18 26 W Data used in comparisons.

18 27 W Data used in comparisons.

18 29 S Prorating generates detailed information for
doing comparisons.

18 31 A Comparisons necessary for monitoring.

18 32 A Comparisons included in budget reports.

18 33 A Comparisons included in periodic analysis
reports.

18 34 A Some comparisons included in summary reports.

18 39 A May include comparisons.

18 41 A May include comparisons.

20 22 A Related data maintenance issue.

20 23 A Related data maintenance issue.

20 24 A Related data maintenance issue.

20 25 A Related data maintenance issue.

20 31 A Data used in monitoring.
 20 32 A Data is source for reports.
 20 33 A Data is source for reports.
 20 34 A Data is source for reports.
 20 35 A Data is source for reports.
 20 36 A Data is source for reports.
 20 37 W Data possibly source for reports.
 20 41 A Data possibly source for reports.
 20 58 A Chart data used in checking.
 20 62 W Type of supplementary data.
 20 66 W Type of supplementary data.
 20 68 W Requires certain supplementary Chart data.
 20 72 A Descriptors may need to be used for
 supplementary Chart data.
 20 73 W Descriptors may need to refer to
 supplementary Chart data.
 21 23 W Would need to transfer data at year end.
 21 29 S Requires current budget data.
 21 30 S Used in monitoring reports.
 21 31 S Used in monitoring reports.
 21 32 S Used in monitoring reports.
 21 33 A Used in monitoring reports.
 21 35 W Portions of data may be used as input to DM.
 21 38 A Need budget data to do function.
 21 39 A Choices may include CFY budget data.
 21 40 S Variations may require certain CFY budget

data.

- 21 60 A Proposed revire often requires examination
of CFY budget data.
- 21 62 A Part of CFY budget data.
- 22 29 A Future budget data submitted in prorated
form subject to change.
- 22 32 S Data needed for reports.
- 22 35 A Data needed by EM.
- 22 36 A Requests for future budget data.
- 22 41 A May require future budget data.
- 22 60 A Review of data maintained.
- 22 61 A Review of data maintained.
- 22 62 W S of F data must be maintained with future
budget data.
- 22 63 A Automatic changes will be applied to future
budget data.
- 22 65 A Commonality in how to treat data.
- 23 36 A Needed for requests.
- 24 31 A Needed for monitoring operating budget.
- 24 33 A Source data for reports.
- 24 34 A Source data for reports.
- 24 36 S Heavily used data for ad hoc requests of
information.
- 24 39 A Optional data items must be available or
derivable.
- 24 40 A Required data for variations may include

current actual data.

- 24 41 A May include current year actual data.
- 24 58 A Needed for editing and checking of future budgets.
- 25 30 W Actual data needed for funds application.
- 25 31 W Needed for comparison to last year.
- 25 32 A LFY Actual data is potential data source.
- 25 33 A LFY Actual data is potential data source.
- 25 34 A LFY Actual data is potential data source.
- 25 35 A LFY Actual data is potential data source.
- 25 36 A LFY Actual data is potential data source.
- 25 39 A LFY Actual data is potential data source.
- 25 40 W LFY Actual data is potential data source.
- 25 41 A LFY Actual data is potential data source.
- 26 36 W Ad hoc requests may need old budget data.
- 27 36 W Ad hoc requests may need old budget data.
- 27 53 A DM may need actual data for extrapolation/modelling.
- 28 38 A Hiring data not now maintained.
- 28 56 W Manual preparation may make use of some data.
- 28 57 S On-line preparation will need data.
- 28 58 A Certain preparation data needed for checking/editing.
- 28 59 W Preparer might need to interact with limits information.
- 28 60 A May need special data for review.

- 28 61 A May need data for changes review.
- 29 30 S Effective use of fund accounts depends on accurately prorated data.
- 29 31 A Prorated data important for operations monitoring.
- 29 32 A Prorating used in FB report generation.
- 29 40 W Variation may involve modification to proration arrangements.
- 29 49 W Changing proration may be done directly by managers via on-line access.
- 29 56 A Budget prorated when proration established.
- 29 57 A Budget prorated when proration established.
- 29 63 A Automatic change to proration also.
- 30 36 A Ad hoc requests for information regarding fund accounts important.
- 30 37 S Fund reports should be addressed toward effective use.
- 30 62 W Easy availability of S of F information should improve effectiveness of use.
- 30 69 A Makes for more effective use.
- 30 70 W Drafts related to effectiveness of use.
- 30 71 A Fund draft checking can intercept ineffective use.
- 31 32 S Periodic budget-versus-actual reports serve monitoring needs.
- 31 33 A Periodic reports may serve monitoring

capability.

- 31 34 A Summary monitoring reports.
- 31 36 W Ad hoc monitoring.
- 31 41 W May need to develop new monitoring reports.
- 31 61 A Changes impact monitoring function.
- 31 68 W Special items may need special monitoring.
- 32 33 W Common reporting issues.
- 32 34 W Common reporting issues.
- 32 37 W Common reporting issues.
- 32 40 A Related reporting issue.
- 32 41 A Related reporting issues.
- 32 61 W Changes may be reviewed via reports.
- 32 64 W Needed for certain report requirements.
- 33 34 A Common data source.
- 33 36 A Common data for typical queries and reports.
- 33 37 W Common data source.
- 33 39 S Common report modification issue.
- 33 40 A Common data and reports.
- 33 41 S Monthly analysis scheme common.
- 33 62 W Sources of funding information may be in
some reports.
- 34 36 S May produce special-purpose report using ad
hoc facility.
- 34 41 W May develop new budget reports out of
special-purpose reports.
- 34 42 A May produce some special reports using

report writer.

36 41 W Ad hoc requirements may lead to new reports.

36 42 S Most ad hoc requirements accommodated via
report writer.

36 43 A Users may wish to develop their own reports.

36 45 A May use query facility to answer ad hoc
questions.

36 48 S Data control makes ad hoc query use more
difficult.

36 72 A New data item types make ad hoc requirements
easier to implement.

37 62 A Funding sources will be shown in funds
report.

39 40 A May want general mechanisms for modification.

39 41 W Modifications may lead to new reports.

39 72 W New data item types can carry data regarding
report content.

40 41 A Variations may lead to new reports.

40 65 W Budget report format may be used for
research proposal reports.

41 65 A Research proposal information prime
candidates for new reporting.

41 72 A Common data organization issue.

42 43 S Facility intended for users.

42 46 W Destination code in reports.

43 46 A Report directing aids must be available to

all users and easy to use.

43 47 W Need to log id information regarding report
writer users.

43 48 W Constrains use of report writer.

43 74 S Train use of report writer - make it easy to
use and learn.

44 46 W Possibly need to dump menu-query output to
terminals.

44 47 A Log query users.

44 48 A Control query access.

44 60 A Proposal review could use query facility.

44 61 A Proposal review could use query facility.

44 71 A Use query facility for funds drafts.

44 74 A User training would be necessary.

45 46 W May need to send ad hoc results to users
terminals.

45 47 A Log query users.

45 48 A Control query access.

45 74 A Need user training.

46 77 W Can achieve faster distributino via remote
printing of reports.

47 48 A Need identity to determine permissable users.

47 49 A Need identity to determine owner.

47 53 A Activity logging would include database
changes.

48 49 A Common access control issue.

48 50 S Common access control.
 48 52 A Encoding can be used to insure that only
 permissible users get access.
 49 51 S Need owner id concept to implement control.
 49 52 W Encoding can be used to insure that only
 permissible users can change items.
 49 53 W Validity of change attempts logged.
 50 51 W Common access issue.
 53 55 S Need log to restore.
 53 61 A Can locate changes via log stream. .
 53 63 W Need to log automatic changes to EBS.
 53 67 S Log is important part of audit trail.
 53 72 A Need to log additions of new data elements.
 54 55 S Restore via backup.
 54 67 W Might want to perform control total checks
 prior to backup.
 55 67 A Restoration would have to meet audit checks.
 56 57 W Common prep. activities.
 56 58 W Data format, etc., common.
 56 60 W Entry of manual proposals must lead to
 Budget Office review.
 56 74 W Need documentation for manual entry.
 57 58 A Will want to check on-line preparation.
 57 60 W Common on-line processing.
 57 74 W On-line activities for users must be easy to
 learn and teach.

58 59 S Control limits key to automatic checking.

58 60 A Budget Office review forms part of check.

58 62 A Check sources as part of checking procedures.

58 66 W Check that category codes correct.

58 75 A Adds complexity to automatic checking
function.

59 60 A Review against controlling limits.

59 61 A Review changes controlling limits.

59 62 A Controlling limits may be tied to S of F.

59 63 W Automatic incrementation in employee
benefits requires secondary check.

59 65 W Suspense file approach tied in with
controlling limits.

59 68 W There are limits on such items.

59 71 S Commonality in performing checks.

59 75 A Control limits must take this into account.

59 76 A Relevant to checking overhead amounts.

60 61 A Common review mechanisms.

60 62 S Sources of funding important aspect of
review.

60 65 A OSP can use same mechanism to review
research proposals.

60 66 W Would have to check categories.

60 68 W Checking overlap, although mainly manual.

60 71 A Budget Office must be able to review these
also - common facility possible.

- 61 62 A Changes often incorporate new funding.
- 61 63 W Automatic changes make change review more
difficult to execute.
- 61 65 A May need to review FP changes.
- 61 66 W May need to review FP changes.
- 61 68 A Changes to budgeted amounts of some
non-recurring expenses may need to be
reviewed.
- 61 71 A Common review requirements - potential
common facility.
- 62 65 W S of F data related to research proposal
preparation.
- 62 71 A Need to check fund draft against account
funded to.
- 63 67 A Need to keep audit log of changes to
employer benefits.
- 63 75 S Automatic changes more difficult under this
requirement.
- 64 74 A Need to be able to train in all techniques.
- 65 66 A Categorization information part of data base.
- 65 76 A Overhead rates are important in manipulation
of proposals.
- 67 68 A Non-recurring expense transactions must be
included in audit trail.
- 67 69 A Fund drafts must be included in audit trail.
- 67 70 A Fund drafts must be included in audit trail.

67 75 W Non-standard employee benefits must be
auditable.

67 76 W Overhead recovery rates must be auditable.

68 71 W Fund drafts for most non-recurring expenses
must be checked.

69 70 A Common processing issue.

72 73 S Application of new data item types.

Appendix L

REQUIREMENT SUBSETS DERIVED FROM THE BEST SYSTEM
DECOMPOSITION.

*** SUBPROBLEM 1 ***

7. Data regarding special financial arrangements made by managers will be maintained.
28. Data required for budget proposal preparation and not available elsewhere will be maintained.
38. The system will have access to certain personnel hiring data.
56. Budget proposals can be prepared manually.
57. Budget proposals can be prepared on-line.
58. Budget proposals will be automatically checked and edited to the extent possible.
59. There will be controlling limits on amounts in proposed budgets and budget changes.
60. Budget proposals will be reviewed by the budget office and/or OSP.
61. Budget changes will be reviewed by the budget office and/or OSP.
62. Sources of funding for each account will be identified.
65. Research proposal information can be stored, updated, and accessed.
66. Research accounts and proposals will be categorized by type, discipline, and function.
68. Required aspects of non-recurring expenses will be supported.
71. Fund drafts can be checked and approved on-line by Budget Office.

76. Overhead recovery rate calculations will be supported.

***** SUBPROBLEM 2 *****

18. There will be a general comparison reporting capability.
19. Items exceeding prespecified bounds can be highlighted.
20. Chart of Accounts and associated supplementary data will be accessible by Budget System.
21. Current fiscal year budget data will be maintained.
22. Future Budget data will be maintained.
23. Last fiscal year Budget data will be maintained.
24. Current fiscal year Actual data will be maintained.
25. Last fiscal year Actual data will be maintained.
26. Historical Budget data will be maintained for up to 1 years.
29. Budgeted income and expense data will be prorated automatically on a month-by-month basis.
31. Monitoring of the operating budget will be supported.
32. Various Future Budget reports will be generated.
33. Periodic operating reports for each account will be generated.
34. Various special-purpose (non-periodic) reports will be generated.
36. Ad hoc requests for information will be supported.
39. Certain budget report data items can be optionally included or excluded from Monthly Analysis Report

as specified by the user.

- 40. Variations of standard Budget System reports will be developed as requested.
- 41. New budget reports will be developed as required.
- 42. There will be a simple report-writing facility.

*** SUBPROBLEM 3 ***

- 16. Budgeting and planning data can be accessed readily by other systems.
- 43. Users can develop their own special reports using the report writing facility.
- 44. There will be an on-line menu-oriented query facility.
- 45. There will be an on-line ad hoc query facility.
- 46. Reports generated on-line can be directed to the system printer or to users' printer/terminal.
- 47. Identity and activity of users will be logged.
- 48. Data items can be accessed only by permissible users.
- 49. Data items can be changed only by their owner.
- 50. There will be a set of permissible users for each data item.
- 51. Each data item will have a unique owner.
- 52. Data items can be encoded for security.
- 64. All currently used budgeting techniques will be supported.
- 74. Formal training in the use of the system will be provided to users.

***** SUBPROBLEM 4 *****

- 15. Reports will be physically easy to handle.
- 77. Standard periodic reports will be produced and distributed to users as quickly as possible.

***** SUBPROBLEM 5 *****

- 9. Personnel budgets can be developed in EFT.
- 10. There will be a facility for converting salary/wage information between dollars and EFT.
- 13. Manpower can be reported in alternative units.

***** SUBPROBLEM 6 *****

- 53. There will be a log of all database changes.
- 54. There will be a file backup facility.
- 55. The database's integrity can be restored using backup files and change log if necessary.
- 67. There will be documented audit trails for determining system discrepancies.
- 69. Fund drafts can be performed on-line.
- 70. Fund drafts can be performed in batch mode.

***** SUBPROBLEM 7 *****

- 11. Certain objects may be logically related to groups of other objects.
- 12. A particular object may belong to multiple object groups.

14. There will be special object codes available for use by each department.

*** SUBPROBLEM 8 ***

5. Short-term and long-term planning data will be provided by Senior Managers.
6. Planning data will be provided in a standardized format.
27. Historical Actual data will be maintained for up to 1 years.
35. The data necessary for use by the Dynamic Model will be generated.

*** SUBPROBLEM 9 ***

8. Personnel budgets can be developed in dollars.
63. Budgeted employee benefits will be changed automatically following changes in budgeted salary and wage amounts or EB rates.
75. Non-standard employee benefit calculations will be supported.

*** SUBPROBLEM 10 ***

1. Fund accounts will be categorized by fund purposes.
2. Fund accounts can be categorized by principal type.
3. Fund accounts can be categorized by income type.
4. Each fund account will be described via abbreviated text description.

- 17. The budget system can access directly data in other systems' files.
- 30. Information to facilitate the effective use of fund accounts will be available.
- 37. Various funds reports will be generated.

***** SUBPROBLEM 11 *****

- 72. Additional data item types can be added to accounts or objects within accounts.
- 73. New data items can include or refer to supplementary data needed by users.

Appendix M

INTERDEPENDENCIES BETWEEN REQUIREMENT SUBSETS IN BEST DECOMPOSITION.

CLUSTER PAIR	INTERDEPENDENT NODES		
1, 2	38, 21,A 58, 24,A 61, 22,A 62, 20,W 62, 29,A 65, 40,W 68, 20,W	56, 29,A 60, 21,A 61, 31,A 62, 21,A 62, 33,W 65, 41,A 68, 31,W	58, 20,A 60, 22,A 61, 32,W 62, 22,W 65, 22,A 66, 20,W
1, 3	56, 74,W 61, 44,A	57, 74,W 71, 44,A	60, 44,A
1, 4	** NONE **		
1, 5	28, 10,A 59, 9,W	38, 9,W	38, 10,W
1, 6	61, 53,A	68, 67,A	76, 67,W
1, 7	** NONE **		
1, 8	** NONE **		
1, 9	58, 75,A 61, 63,W	59, 63,W	59, 75,A
1, 10	7, 3,A 58, 1,A 62, 37,A 71, 30,A	7, 30,A 60, 1,A 71, 1,W	7, 37,W 62, 30,W 71, 17,W
1, 11	** NONE **		
2, 3	29, 48,W 36, 45,A 42, 46,W	32, 64,W 36, 48,S	36, 43,A 42, 43,S
2, 4	** NONE **		
2, 5	29, 9,A	33, 9,A	

2, 6	** NONE **		
2, 7	34, 11,A	34, 12,S	
2, 8	18, 27,W 22, 5,S 25, 35,A 36, 27,W	20, 35,A 22, 6,W 34, 5,A	21, 35,W 22, 35,A 34, 6,W
2, 9	22, 63,A	29, 8,A	
2, 10	19, 3,A 20, 3,A 21, 30,S 32, 37,W 36, 2,W 36, 30,A	20, 1,A 20, 17,S 25, 30,W 33, 37,W 36, 3,A	20, 2,A 20, 37,W 29, 30,S 36, 1,W 36, 4,W
2, 11	20, 72,A 39, 72,W	20, 73,W 41, 72,A	36, 72,A
3, 4	46, 77,W		
3, 5	** NONE **		
3, 6	47, 53,A	49, 53,W	
3, 7	** NONE **		
3, 8	** NONE **		
3, 9	** NONE **		
3, 10	** NONE **		
3, 11	** NONE **		
4, 5	** NONE **		
4, 6	** NONE **		
4, 7	** NONE **		
4, 8	** NONE **		
4, 9	** NONE **		
4, 10	** NONE **		
4, 11	** NONE **		
5, 6	** NONE **		
5, 7	** NONE **		
5, 8	** NONE **		

5, 9	10, 8,S	13, 8,A	
5, 10	** NONE **		
5, 11	** NONE **		
6, 7	** NONE **		
6, 8	** NONE **		
6, 9	53, 63,W	67, 63,A	67, 75,W
6, 10	69, 17,W	69, 30,A	70, 30,W
6, 11	53, 72,A		
7, 8	11, 35,A		
7, 9	** NONE **		
7, 10	** NONE **		
7, 11	** NONE **		
8, 9	** NONE **		
8, 10	** NONE **		
8, 11	** NONE **		
9, 10	** NONE **		
9, 11	** NONE **		
10, 11	1, 72,S		

BIOGRAPHICAL NOTE

Sidney L. Huff was born in Rochester, New York, on June 14, 1945. He attended primary and a portion of secondary school in the Rochester area. He emigrated to Oakville, Ontario, Canada in 1961, and completed high school in Oakville.

He attended Queen's University, Kingston, graduating with a B.Sc. (Honors) in Applied Mathematics in 1968, a M.Sc. in Electrical Engineering in 1970, and a Masters of Business Administration in 1972. During this time he also worked as a research engineer for Ontario Hydro Research Laboratories in Toronto, and in various research positions at Queen's University.

In 1972 he joined the Canadian management consulting firm Woods, Gordon & Co., in the Management Engineering group. While there he completed a number of consulting assignments in various parts of Canada. In 1973 he accepted a faculty position in the School of Business, Queen's University. He taught there, primarily in the managerial accounting, management information systems, and management science areas, for two years prior to beginning his doctoral studies.

He began his doctoral program at the Sloan School of Management, M.I.T., in the fall of 1975, majoring in Management Information Systems. While at M.I.T. he taught the course 15.561 (Benchmark Computer Programming) in 1976, and served for four years on the doctoral program committee.

Mr. Huff has accepted a position as Assistant Professor at the University of Western Ontario, in London Ontario, where he will teach in the Management Science and Management Information Systems fields.